# TUGboat

Volume 44, Number 1 / 2023

**TEX Users Group**

*TUGboat* (ISSN 0896-3207) is published by the TEX Users Group. Web: `tug.org/TUGboat`.

**Individual memberships**
2023 dues for individual members are as follows:
- Trial rate for new members: $30.
- Regular members: $105.
- Special rate: $75.

The special rate is available to students, seniors, and citizens of countries with modest economies, as detailed on our web site. Members may also choose to receive *TUGboat* and other benefits electronically, at a discount. All membership options are described at `tug.org/join`.

Membership in the TEX Users Group is for the calendar year, and includes all issues of *TUGboat* for the year in which membership begins or is renewed, as well as software distributions and other benefits. Individual membership carries with it such rights and responsibilities as voting in TUG elections. All the details are on the TUG web site.

**Journal subscriptions**
*TUGboat* subscriptions (non-voting) are available to libraries and other organizations or individuals for whom memberships are either not appropriate or desired. Subscriptions are delivered on a calendar year basis. The subscription rate for 2023 is $115.

**Institutional memberships**
Institutional membership is primarily a means of showing continuing interest in and support for TEX and TUG. It also provides a discounted membership rate, site-wide electronic access, and other benefits. For further information, see `tug.org/instmem` or contact the TUG office.

**Trademarks**
Many trademarked names appear in the pages of *TUGboat*. If there is any question about whether a name is or is not a trademark, prudence dictates that it should be treated as if it is.

[printing date: April 2023]

Printed in U.S.A.

**Addresses**
TEX Users Group
P. O. Box 2311
Portland, OR 97208-2311
U.S.A.

**Telephone**
+1 503 223-9994

**Fax**
+1 815 301-3568

**Web**
`tug.org`
`tug.org/TUGboat`

**Electronic mail**
General correspondence, membership, subscriptions:
`office@tug.org`

Submissions to *TUGboat*, letters to the Editor:
`TUGboat@tug.org`

TEXnical support, public mailing list:
`support@tug.org`

Contact the Board of Directors:
`board@tug.org`

In 1758, the British mathematician Francis Maseres professed that negative numbers:
"… darken the very whole doctrines of the equations and make dark of the things which are in their nature excessively obvious and simple."

Deyan Ginev
email to the W3C MathML Working Group, 26 May 2022

# TUGboat

COMMUNICATIONS OF THE TeX USERS GROUP
Editor Barbara Beeton

Volume 44, Number 1, 2023
Portland, Oregon, U.S.A.

**_TUGboat_ editorial information**

This regular issue (Vol. 44, No. 1) is the first issue of the 2023 volume year. The deadline for the second issue in Vol. 44 (the TUG'23 conference proceedings) is July 23, 2023, and for the third (regular) issue, October 15, 2023. Contributions are requested.

*TUGboat* is distributed as a benefit of membership to all current TUG members. It is also available to non-members in printed form through the TUG store (`tug.org/store`), and online at the *TUGboat* web site (`tug.org/TUGboat`). Online publication to non-members is delayed for one issue, to give members the benefit of early access.

Submissions to *TUGboat* are reviewed by volunteers and checked by the Editor before publication. However, the authors are assumed to be the experts. Questions regarding content or accuracy should therefore be directed to the authors, with an information copy to the Editor.

**_TUGboat_ advertising**

For advertising rates and information, including consultant listings, contact the TUG office, or see:
`tug.org/TUGboat/advertising.html`
`tug.org/consultants.html`

**Submitting items for publication**

Proposals and requests for *TUGboat* articles are gratefully received. Please submit contributions by electronic mail to `TUGboat@tug.org`.

The *TUGboat* style files, for use with `plain` TeX and LaTeX, are available from CTAN and the *TUGboat* web site, and are included in TeX distributions. We also accept submissions using ConTeXt. For deadlines, templates, author tips, and more, see `tug.org/TUGboat`.

Effective with the 2005 volume year, submission of a new manuscript implies permission to publish the article, if accepted, on the *TUGboat* web site, as well as in print. Thus, the physical address you provide in the manuscript will also be available online. If you have any reservations about posting online, please notify the editors at the time of submission and we will be happy to make suitable arrangements.

**Other TUG publications**

TUG is interested in considering additional manuscripts for publication, such as manuals, instructional materials, documentation, or works on any other topic that might be useful to the TeX community in general.

If you have such items or know of any that you would like considered for publication, please contact the Publications Committee at `tug-pub@tug.org`.

## TUG Institutional Members

TUG institutional members receive a discount on multiple memberships, site-wide electronic access, and other benefits:
`tug.org/instmem.html`
Thanks to all for their support!

American Mathematical Society, *Providence, Rhode Island,* `ams.org`

Association for Computing Machinery, *New York, New York,* `acm.org`

Aware Software, *Newark, Delaware,* `awaresw.com`

Center for Computing Sciences, *Bowie, Maryland*

CSTUG, *Praha, Czech Republic,* `cstug.cz`

CTAN, `ctan.org`

Duke University Press, *Durham, North Carolina,* `dukeupress.edu`

Hindawi Foundation, *London, UK,* `hindawi.org`

Institute for Defense Analyses, Center for Communications Research, *Princeton, New Jersey*

L3Harris, *Melbourne, Florida,* `l3harris.com`

LaTeX Project, `latex-project.org`

MacTeX, `tug.org/mactex`

Maluhy & Co., *São Paulo, Brazil,* `maluhy.com.br`

Marquette University, *Milwaukee, Wisconsin,* `marquette.edu`

Masaryk University, Faculty of Informatics, *Brno, Czech Republic,* `fi.muni.cz`

Nagwa Limited, *Windsor, UK,* `nagwa.com`

Overleaf, *London, UK,* `overleaf.com`

StackExchange, *New York City, New York,* `tex.stackexchange.com`

TeXFolio, *Trivandrum, India,* `texfolio.org`

Université Laval, *Ste-Foy, Québec, Canada,* `bibl.ulaval.ca`

University of Ontario, Institute of Technology, *Oshawa, Ontario, Canada,* `ontariotechu.ca`

University of Oslo, Institute of Informatics, *Blindern, Oslo, Norway,* `uio.no`

VTeX UAB, *Vilnius, Lithuania,* `vtex.lt`

## From the president

Boris Veytsman

It is not often one is chided by Don Knuth, but I got this distinction. In the interview with Paulo Ney I mentioned the program I've called *Maxima*. DEK wrote to *TUGboat*,

> I'm writing today just to mention a glitch that I noticed. The transcript of Paulo's excellent interview with Boris refers several times on page 103 to a software system called "Maxima", which Richard Fateman had brought to Berkeley.
>
> No; that program was called "Macsyma". It probably has some historic connection with the system Boris calls "Maxima" on page 133, reference [10]; but page 103 (and also page 104) certainly was not about Joel Moses's pioneering Macsyma system.

Of course, DEK is right: the system Richard Fateman brought to Berkeley is properly called *Macsyma*. However, the story behind the name is rather complex. It is mentioned on the *Maxima* pages, `maxima.sourceforge.io`. When *Macsyma* was being developed (1960s), our ideas about free software and its licensing were not as clear as today. Thus while the code became available, its license was not free. I started to use the program in the mid-1990s, when it was consecutively sold by several commercial companies, still under the name *Macsyma*. It became one of my favorite tools (I think somewhere in the archives one can still find my bug reports). Unfortunately the companies folded, and the future of this wonderful program, with many thousands of human hours spent in its development, became questionable. In a stroke of good luck, William Schelter, who had maintained *Macsyma* since 1982, was able after years of lobbying to secure the permission of DOE to release the code under GPL. The property rights on the name *Macsyma* being unclear, the program was released as *Maxima*, and this is how it is known today. Thus *Maxima* developers state that *Maxima is simply the most recent name for the branch that started under the name DOE* Macsyma (`maxima.sourceforge.io/faq.html`). Richard Fateman seems to agree with this definition, mentioning about his projects *[o]ne for which I've contributed is the (now public version) of the* Macsyma *program, named* Maxima. (`people.eecs.berkeley.edu/~fateman/`) Thus while the program Richard Fateman brought to Berkeley was *Macsyma*, its current public version is, due to legal complications, *Maxima*.

The reason I have spent some time dwelling on this story is that we now understand something which was not clear decades ago: our code may have a longer life than we envision when we write it. When I worked at NASA Goddard Space Flight Center a quarter century ago, I supported a program written before my birth. I have been told it is still in use. *Maxima* itself, despite its age, is very popular. I use it almost daily. Recently I compared its performance with *ChatGPT* on symbolic calculation of integrals. While *Maxima* always gave me correct answers, *ChatGPT* did not. After it produced another wrong result, I tried to help the computer, and said *I think this integral is* $\pi$*.* *ChatGPT* answered, *I apologize for my mistake earlier. You are correct, the value of the integral is actually* $\pi$*.* Then it produced another page of pompous calculations, ending with a flourish, *so the overall value of the integral is* $4 \cdot 6 + 2 \cdot 1 = \boxed{\pi}$ (the final emphasis is by the computer). What a contrast with the modest and completely correct output of *Maxima*!

It seems we are going to use the code base of *Maxima* for a long time. We are fortunate that due to the efforts by William Schelter its code is no longer under a proprietary license.

We are even more fortunate that DEK understood these issues long ago, and TEX has been free software from the beginning. This enabled the community to create this beautiful set of programs commonly referred to as *TEX and friends*. Many of us in the community spend our time maintaining and extending them as developers, users, or serving in a users group.

Which brings me to the last topic of this column. Since 2017 I have had the privilege to support TEX as the President of TUG. This year I am stepping down. Many thanks to Arthur Rosendahl, who decided to take on this duty. It has been a very interesting journey, and I am grateful to the community that trusted me with this office all these years.

One of the most important duties of the President is, in my opinion, writing the letters to the community: monthly newsletters and columns in *TUGboat*. At this time I have written 66 newsletters and 11 columns, this one being the last (I still have some newsletters to write until Arthur assumes office). Rereading them, I feel many were naïve, some were written in haste, and all could be improved. I can only sum up by saying they were written without malice, and to the best of my abilities.

*Happy TEXing,*

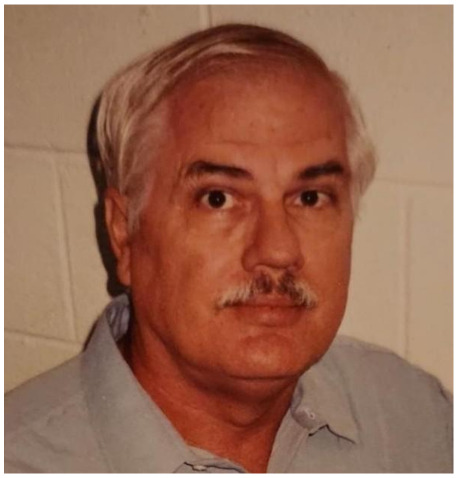
◇ Boris Veytsman
`president (at) tug dot org`

## Editorial comments

Barbara Beeton

### Donald P. Story, 1946–2022

Don Story, an active participant in the LaTeX community, passed away on the last day of 2022, after a long battle with ALS. A member of the Mathematics faculty of the University of Akron from 1976 until his retirement in May 2006, Don's interest in TeX was directed mainly toward helping others communicate mathematics. His presence on CTAN consists of an astounding 62 packages. The most well known of these is `acrotex`, the AcroTeX education bundle. This bundle provides tools based on cooperation between Adobe Acrobat and TeX "for putting mathematics on the internet".[1] Its `README` file shows activity well into 2021, and ends with the note "Now, I simply must get back to my retirement."[2]



Born 17 December 1946 in El Paso, Texas, Don didn't stay there for long. As an "Air Force brat", he moved around with his family for 19 years, mostly in Europe, but ended up at the University of Florida where he received his Ph.D. in mathematics. In addition to his position at the University of Akron, for a time he worked for Adobe, hence his solid knowledge of Acrobat. After retirement, he moved to Florida, where he continued to mentor students and develop his packages at Northwest Florida State College.

Don's death was reported to us by his friend and colleague Tom Price, Professor Emeritus, Mathematics, at the University of Akron. Tom also reported that, in their younger years, he and Don were avid table tennis opponents and their friendship persisted even though Don invariably won.

[1] AcroTeX: Acrobat and TeX Team Up, *TUGboat* **29**:3 (1999), pp. 196–201.

[2] `ctan.org/pkg/acrotex`

### Clarification: Interview with Boris Veytsman, *TUGboat* 43:2

In the subject interview from the TUG 2022 proceedings, the software "Maxima" is referred to several times on pages 103–104. The references begin with this statement by Paulo Ney de Souza, the interviewer:

> I had just started at the time playing with Maxima here and Richard Fateman had brought the Maxima code and had installed it on the VAXes at Berkeley.

Don Knuth, in a message to me, has pointed out that the system brought by Fateman to Berkeley was called "Macsyma", and further says that

> It probably has some historic connection with the system Boris calls "Maxima" on page 133, reference [10]; but page 103 (and also page 104) certainly was not about Joel Moses's pioneering Macsyma system.

The confusion is cleared up in the section entitled "History" on the referenced page (`maxima.sourceforge.io/`):

> Maxima is a descendant of Macsyma, the legendary computer algebra system developed in the late 1960s at the Massachusetts Institute of Technology. It is the only system based on that effort still publicly available and with an active user community, thanks to its open source nature.

So the software referred to by Paulo is "Macsyma", and later references by Boris are to its open source descendant "Maxima".

### News from the TeX neighborhood

As expected, the TeX Live 2023 pretest was launched at the beginning of the year. TeX Live 2022 was frozen just before the release of the new edition on March 19. Information regarding the release is on the TUG web pages, at `tug.org/texlive`; the files are also on CTAN. There are a few notable changes:

- Windows binaries are now 64-bit (not 32), and are thus installed in the directory `bin/windows` instead of `bin/win32`.
- The version of ConTeXt is LMTX, based on the new LuaMetaTeX engine by Hans Hagen; the MkII format based on pdfTeX is no longer included (though it may return in some other guise). Binaries are not provided for all possible platforms, but a snapshot of how to build them is present in the TL `source` tree.

- MacTEX includes the `hintview` viewer for Martin Ruckert's `hitex` engine/`hint` output files; Martin is hoping for useful feedback.

The release has been sent for DVD production, and the DVDs should be ready to start mailing by the end of May.

This year's LATEX update is expected by the end of June. The third edition of *The LATEX Companion*, now in two volumes, is available for order; it will be published in both print and ebook formats on May 15, 2023. If you start from the links on `latex-project.org` or `tug.org`, you will get a discount and the authors will get a small extra commission.

For users of MetaPost, GUTenberg has established a site devoted to its uses and examples: `metapost.gutenberg-asso.fr`.

The *really* big news is that, after several years of enforced isolation, this year's TUG meeting will finally, once again, be held in person. It will take place in Bonn, Germany, the former capital of West Germany, and Beethoven's birthplace, from July 14–16, with a developer's workshop on "Tagging PDF documents" the day before. Many thanks are due to Ulrike and Gert Fischer, residents of Bonn, who have volunteered to make the local arrangements. By the time you read this, the dates for obtaining rooms at the conference hotel and for proposing a talk will have passed, but it should still be possible to register to attend. Check the conference web page (`tug.org/tug2023`). It's hoped that it will be possible to also make the conference sessions available online, although the schedule will be more "traditional" than it has been during the COVID lockdown, taking place, most likely, from 9 a.m. to 5 p.m. Central European time.

Sadly, the retirement of DANTE's long-time office staff and the difficulty of finding a dedicated replacement has resulted in the inability of DANTE to hold their accustomed Spring meeting. Instead, a brief meeting will be held in Bonn on July 13, prior to the TUG meeting.

A new version of the Lucida font collection has been released. The most significant changes affect the math fonts. Details are reported at `tug.org/pipermail/lucida/2023-January/000921.html`. and in a *TUGboat* article by Hans Hagen and Mikael Sundquist.[3] There's no additional charge for anyone who has already purchased a license.

A potential "alternative" to LATEX has appeared: Typst (`typst.app`). Developed by two former students of the Technical University of Berlin who "were unhappy with LATEX and how slow and unwieldy it was", they undertook to create a replacement written in Rust that would be easier to use while producing output of comparable quality. Not a backslash in sight. A user-focused talk on Typst has been submitted to TUG'23.

The introduction to LATEX, "Formatting Information" by Peter Flynn, first introduced here in *TUGboat*,[4] has been updated to incorporate recent developments. This online book, now in its 8th edition, is mainly for beginners. The new edition can be found at `http://latex.silmaril.ie/`. Peter requests that bugs, comments, and suggestions be sent to `latex@silmaril.ie`.

---

## Accessibility for (LA)TEX

Accessibility has become a more and more pressing concern. The LATEX Team and other members of the TEX community have devoted considerable attention to the problem,[5] as has the W3C MathML Working Group. To the best of my knowledge, not much effort has been made to bring this to the notice of the ordinary scientist, but an article, "Making Accessible Documents Using LATEX" in the January issue of the AMS *Notices*[6] made a start at remedying that situation. Whether this will result in the desirable changes required from individual authors to make the mathematical literature readily accessible to those readers capable of understanding it or trying to learn the subject matter, is still an unanswerable question, but at least it's emerging from the shadows.

A website, Accessible Mathematics,[7] created and maintained by Abbas Jaffary, presents a comprehensive overview of the problems associated with rendering mathematics accurately, whether spoken, in Braille, or online in visible, non-PDF form. Notation can be ambiguous, depending on context for the exact meaning; relevant examples indicate the extent of the problem. This website addresses not only (LA)TEX, but also many other tools used to record math in electronic form. Included is MathML, which is also being examined in detail for the same purpose.

MathML is the subset of XML dealing specifically with mathematical notation. For several years, the MathML Working Group has been addressing the issues involved in converting files containing MathML code to forms other than publication on paper or as PDF. A mailing list adjunct to their "internal"

[3] `tug.org/TUGboat/tb43-3/tb135hagen-lucida.pdf`

[4] `tug.org/TUGboat/tb23-2/tb74flynn.pdf`

[5] See `tug.org/twg/accessibility/`

[6] `www.ams.org/journals/notices/202301/rnoti-p68.pdf`

[7] `www.accessiblemath.info/accessible_math_documentation_page.html`

communications allows other interested persons to eavesdrop on their discussions.[8] The current topic is how to communicate "intent" to the tools used for translation to another form. "Intent" is the information necessary to correctly translate visual expressions such as $|x|$ to a contextually appropriate expression. Although the subject matter of a source may provide enough information to disambiguate such expressions, it may be necessary for the author to provide some direction — and it's not guaranteed that all authors will be willing to make the extra effort.

The arXiv project has issued an accessibility research report, "A framework for improving the accessibility of research papers on `arXiv.org`".[9] This is a good overview of the current state of affairs.

---

## Accessibility — Something completely different

On a tour of the Operations Center for the Chandra X-Ray Observatory (`chandra.si.edu/`), I was surprised to learn about the significant work already accomplished on trying to make the results of astrophysical exploration accessible to those with visual restrictions. Although the wavelengths — short or long — of astronomical objects may be outside the normal range of human vision, the usual adjusted images are still entirely visual.

Two adaptations have been applied to make these images accessible through touch and sound. The resolution of the masses of data collected by Chandra and other orbiting telescopes is exceedingly high, allowing it to be converted to spatial instructions that can drive a 3D printer. The result, for the Cassiopeia A supernova remnant, resembles something spongelike that might be found on a reef, underwater.[10] The Crab Nebula pulsar, on the other hand, rather resembles a piece of exercise equipment — a smooth disc with "handles" emerging from the center on both sides.[11] In the header on the Crab Nebula page, the word "Explore" links to additional examples, all with instructions for setting up a 3D printer to produce the "concrete" object yourself.

"Sonification" is applied to create representations that can be heard. This project was a response to the isolation of the Chandra team members during the coronavirus epidemic, since it allowed them to work while physically separated. The team includes individuals with backgrounds in music and sound engineering as well as astrophysics, and a member of the blind community who is an accessibility expert. Instruments and sonic patterns are chosen to represent elements of an image: gas by drones, individual stars or points of light by struck or plucked instruments, with volume representing brightness, and pitch, the wavelength or color. A selection of these sonic images has been linked as YouTube offerings, with the decisions applying to each image described in clear detail along with the video presentation.[12] A collection of the audio selections will be made available (on vinyl!) with proceeds to go to the Helen Keller Foundation.

Finally, a typographic note: As we entered the suite, we were greeted by an impressive identification spelled out in relief. The font is a relative of "Bank Gothic", designed by Morris Fuller Benton in the early 1930s for American Type Founders (ATF). Elegant and appropriate, in my opinion.

---

## An afternoon at the Museum of Printing

The Museum of Printing (MoP), located in Haverhill, Massachusetts, is a treasure trove of just about everything to do with the printing tradition. (They have also agreed to be the repository for my papers and library dealing with math typography and TeX.)

One event held at MoP in November was a conversation between Frank Romano, president of the museum, and Matthew Carter, type designer extraordinaire. Rather than a formal lecture, this was an informal interchange, sharing common background (Romano and Carter were both, for a while in the 1960s–70s, employed by the Mergenthaler Linotype Corporation in New York City), and exploring Carter's history and various pursuits in the industry.

Born in England shortly before the Second World War, Carter shared some early memories of life during that period. He wasn't aware of his father's profession (book designer and print historian) until his basic education was complete, and he was ready

[8] Send a message to `www-math-request@w3.org` with the subject "subscribe".

[9] `info.arxiv.org/about/accessibility_research_report.html`

[10] A typical photograph, processed by NASA, can be seen at `solarsystem.nasa.gov/resources/822/cassiopeia-a-supernova-remnant/`. The 3D image appears toward the bottom of this page: `chandra.si.edu/deadstar/deadstar.html`.

[11] Near the bottom of the page at `chandra.si.edu/deadstar/crab.html`.

[12] `chandra.si.edu/sound/`

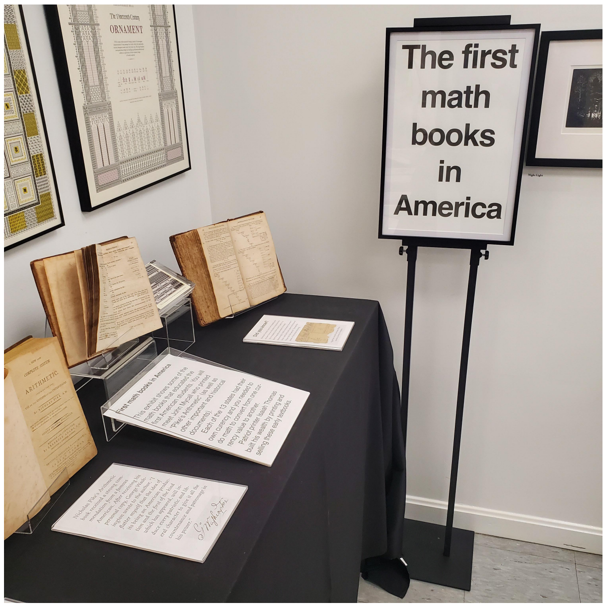
to enter university. But before he headed to Oxford, he spent an intern year at the Enschedé type foundry in Haarlem, where he learned to cut punches, the basis for creating molds from which individual metal types are cast. Having enjoyed this experience, Carter did not enter Oxford as intended (a decision approved by his father), but delved into the world of type design, which at that very moment was moving away from metal to phototype.

After gaining more experience in London, including the use of mylar drafting film as a medium on which to draw letter images, Carter moved to New York in 1960, where he joined Mergenthaler with Mike Parker as a sponsor. Rather than submitting his drawings to the draftsmen who were tasked with creating the reversed-direction formal drawings used for the production of the punches and matrices from which metal type was struck, Carter drew his letters on drafting film in the right-reading direction, from which they were transferred to photographic film using a large graphics camera that was carefully "insulated" from vibration on a granite bench. (The complete run of formal drawings, for every metal font ever produced by Mergenthaler, is in the MoP archives.)

On November 9, 1965, while Carter was in his office at Mergenthaler, all the lights went out. He looked out his window, and there were no lights anywhere, including at the nearby Brooklyn Navy Yard. This was the great East Coast blackout, caused by the failure of a relay at a hydroelectric power plant near Niagara Falls. A cascade of failures along the transmission grid caused the lights to go out over most of the Northeast U.S. and all of Ontario. Carter named the font he was working on at the time "Cascade". (In Rhode Island, I was riding home from the office on a Vespa when the lights went out; it was a memorable experience.)

Until the phototype revolution, Mergenthaler had based its business on hardware. But with phototype, the ability to create fonts became less closely tied to the hardware, and an effort to persuade the company to consider fonts as a separate center of effort failed. Carter and Mike Parker left Mergenthaler to form Bitstream, which specialized in fonts, migrating from phototype to digital type as typesetting methods changed, creating new versions of old familiar fonts and developing new ones both for general use and under commission for such well-known institutions as the *New York Times* and Yale University.

Since 1992, Carter has been associated with Carter & Cone. He is still actively designing type, an occupation which seems to interest him more than any hobby. A more detailed biography appears online in Wikipedia[13] and the conversation at MoP has been recorded and can be viewed on the MoP channel on YouTube. The transcript, with links, appears later in this issue.



In addition to this event and the permanent displays of presses, related hardware, and their printed products, a topical exhibition at MoP of particular interest to me was a copy of the first math book printed in America: *Arithmetick Vulgar and Decimal*, by Isaac Greenwood, published in Boston in 1729. Along with several volumes and leaves from other early works, a pleasing display occupied a sizeable table.

A complementary item to the Carter-Romano conversation, found quite by chance as an internet posting,[14] celebrates Matthew Carter's Jubilee — 50 years as a type designer. Published in 2005, its text is bilingual, in Czech and English — TYPO: *typografie. graphický design. vizuální komunikace/typography. graphic design. visual communication.* The booklet consists of three main parts: an essay by Margaret Re, "A Typographic Jubilee for Matthew Carter"; an interview with Carter by Adam Twardoch;[15] and an illustrated catalog of the fonts designed by Carter. The interview covers the same territory as the conversation with Romano, perhaps slanted a bit more toward a European audience. Recommended reading.

———

[13] en.wikipedia.org/wiki/Matthew_Carter

[14] twardoch.com/download/TYPO_2005_18_Carter.pdf

[15] Adam Twardoch, an active type designer, is currently president of GUST.

## Arsenic and old books

The Providence Athenæum, a membership library established in 1836, sponsors a wide variety of lectures and salons on the general subject of books. On 24 March 2023, the topic was the presence of poisonous materials in 19th century bookbinding and printing. The presenter, Dr. Melissa Tedone, is Book & Library Conservator at the Winterthur Museum, Garden, and Library in Wilmington, Delaware.

Until the mid-19th century, most dyes and colorants produced rather subdued hues. Beginning in the Victorian era, more exuberant coloring agents became available. These were received avidly by the growing middle class, whose wealth was also increasing and allowed purchases beyond necessities, like books. Unfortunately, many of these new colors incorporated toxic compounds based on heavy metals — arsenic, chrome, lead and mercury.

While restoring a bright green book cover, Dr. Tedone noticed that tiny colored bits were flaking off, leaving a residue on her hands and the work area. She stopped work and took a sample of the residue to the lab for analysis. Tests identified the material as containing copper acetoarsenite, a dangerously toxic compound that exceeded the measuring limit of the lab's test equipment. The book's color is known as "emerald green", the same compound as "Paris green", used to poison rats in the sewers of Paris.

In the 1850s and '60s, for books published in England and the U.S., book cloth was coated with the coloring agent, not dyed. This coating flakes off readily with use. In France and Germany, from the 1830s through the end of the century, the same toxic agents colored the inks printed on paste papers used instead of cloth for covering books. Toxic inks also colored illustrations within books. Another notable use of emerald green was in the colorful wallpaper patterns crated by William Morris.

These toxic compounds were in common use during the period: copper acetoarsenite (brilliant green); mercury (red); chrome and lead (yellow and orange). It's recommended that if you find such a brightly colored 19th century book on your shelves, enclose it in an airtight plastic bag and seek professional guidance.

In-person attendees at the Athenæum received a keepsake, a bookmark printed with green edge strips, one matching the shade typical of books bound in book cloth, and the other matching paste-paper bindings. A video[16] records the simultaneous Zoom presentation, and if you want to obtain a copy of the bookmark, the Winterthur wiki[17] provides instructions for how to do so, as well as a wealth of other information on handling suspect books.

[16] providenceathenaeum.org/media-archives/the-poison-book-project/

[17] http://wiki.winterthur.org/wiki/Poison_Book_Project#Emerald_Green_Color_Swatch_Bookmark

---

## How many TEX documents?

A question from Chuck Bigelow might benefit from wider consideration:

> ... how many mathematics, physics, economics, engineering, chemistry, biology papers have been prepared for publication or distribution (e.g. technical reports, theses, pdfs) with TEX since 1982?
>
> Or from 1979 to 2023 in any version and including forks and branches and other scientific and scholarly disciplines?

Where can we look for relevant information?

I can provide information on the use of TEX by the American Mathematical Society, where TEX has been used for published documents since 1982.

Some publishers of scientific books and journals accept submissions in LATEX, but farm out the files for conversion to XML, which is the archival medium. Elsevier and Springer are in this group.

ArXiv, the repository of papers in math, physics, and other "hard" sciences, reports well over 2 million items in their holdings. They are probably willing to divulge what part of this is in TEX form.

Another source of LATEX document preparation is Overleaf, which is limited strictly to TEX-based activity. They will have a sizeable presence at TUG'23, and I will inquire after their statistics.

Nelson Beebe's growing collection of bibliographies includes several that cover the core TEX-related canon, and other collections for fields that are known heavy users of TEX. Nelson is also expected at TUG'23.

This is a bit of a research project, and suggestions for questions to ask and likely candidates to be asked are welcome. Send your suggestions to me at TUGboat@tug.org. The results will be presented in a followup report.

◇ Barbara Beeton
https://tug.org/TUGboat

# A conversation with type designer Matthew Carter

Frank Romano

This conversation took place on 12 November 2022 at the Museum of Printing, Haverhill, Massachusetts. The interviewer, Frank Romano, is President of the museum. The conversation was recorded in three parts, which can be viewed on YouTube.[1]

**Frank Romano (FR):** We are here with Matthew Carter, one of the few people I know who has his own Wikipedia page. [holds up a printout of the page] I have no written questions. What I've decided to do is essentially go through your Wikipedia page. [both laugh]

**Matthew Carter (MC):** I hope it's accurate.

**FR:** We can find out.

You were born in England, and as you were growing up, it was the time of the blitz in England. Do you have any memory of that?

**MC:** Um, yes! I do remember bomb craters in the street. I remember the contrails of ...

**FR:** German planes?

**MC:** Yeah. I don't remember exactly what they were, but like dogfights. I remember being warned not to pick up any interesting-looking thing that might be in the garden, because it was probably an incendiary bomb. And I remember my grandmother's house having all its windows blown out by a doodlebug[2] which landed in the front yard. Yeah, I remember. I remember the feeling that people were actually trying to kill me and my mother in our own home.

**FR:** Your father was away during most of the war?

**MC:** Most of the war, yes.

**FR:** So when did you realize what he did?

**MC:** What an interesting question. I have *no* idea. My dad was not a very anecdotal person. He didn't bring his work home. When he was home, he was in the garden; he was a keen gardener. So he didn't talk about his work, at all, really. So I guess it was just a gradual process of osmosis. I mean, he had books, of course, at home, which told me what his interests were, and some of them were very interesting to me in time. But I don't remember a sort of lightning-bolt moment of thinking, oh my God, my dad's a typographer. It must have been gradual.

**FR:** I would assume your education was mostly traditional, lower school, reading and writing, ...

**MC:** That's right.

**FR:** What we call high school, and then you went to college.

**MC:** No.

**FR:** So you had no post-secondary education after that.

**MC:** No.

**FR:** You were then apprenticed, to a company in Amsterdam?

**MC:** In Haarlem, in the Netherlands, yeah. I left school in '55, in the summer. And then it's September. I started this, we didn't actually use the term "intern" in those days, but unpaid trainee at Enschedé's. And they had this project, welcoming. John Miles, a very old friend of mine, was the first person, I think, to do that. Then I came, and Carl Dair from Canada, followed. So it was a great scheme that they had, and Enschedé's was a very fascinating place. They're primarily security printers; they print stamps and banknotes and so on, but also general printers. And they had their own type foundry, at least when I was there.

You know, I was supposed to work my way around the whole factory. But I started in the type foundry, and I really sort of stuck there the whole year, working with punchcutters, learning a completely obsolescent trade; obsolete, I should say, trade. But I enjoyed that. It's true, I was supposed to go to university at Oxford, but when I went back after that intermediate year, I couldn't face going back into academic life. I expected a problem with my parents, because my dad was very academic, but, to my relief, they had no problem with my dropping out of Oxford before I began. I guess it saved the fees. [both laugh] So then I kind of started trying to work.

**FR:** So this introduced you to the world of type.

**MC:** Yes.

[1] `www.youtube.com/watch?v=vXFdNLPEC9s`, `www.youtube.com/watch?v=LvOPG5KosiA`, `www.youtube.com/watch?v=k6OMjUhBeDA`

[2] In WWII, this is what the English called the German V1 flying ("buzz") bomb.

doi.org/10.47397/tb/44-1/tb136carter-romano

**FR:** You had no knowledge before that.

**MC:** Well, as I say, my dad had all the books; well, not all the books, but a lot of the books about type, and I was interested enough in that, and at school, I got caught up in the revival of italic handwriting. And indeed, I taught it for a while; this was at my second school, public school as it's called, private school in other words. And so I taught italic handwriting. And so my dad did give me a copy of Edward Johnston's *Writing & Illuminating, & Lettering*; as you know, it's called "the best manual, on any subject, ever written".

**FR:** We have a copy in the library.

**MC:** I'm sure you do. It would be serious if you didn't. [both laugh] So I studied that, as a schoolboy I was interested in that and I drew some lettering for the school magazine and so on, and so, I sort of worked my way into it.

**FR:** I'm just trying to probe how it all began.

**MC:** Yeah.

**FR:** So we have these interesting confluences of you coming together with type. So now you're in Haarlem, and you've learned hot metal. Did you ever make a punch?

**MC:** Oh yes! I did a few. I did some for the University Press project. I did a couple for Fritz Mardersteig, but I don't think he ever used them. I did more than punches. I cut binders' brasses, which is a very similar thing, and there was more demand for that, because, you know, once you've cut the brass, you can use it, but if you cut a punch, you then got to make a matrix. It's a more elaborate process. But yes, I did a number of engraving jobs of one kind or another, but I learned to make type before I could design it, but I quickly learned I had to design it, because commissions cutting punches and brasses and so on were very few and far between.

**FR:** And how many years did you do that?

**MC:** Well, I came back from Holland in '56, and I worked at my parents' home for a couple of years, and then in '58 I moved to London. And in 1960, I came on a visit to this country, which changed my life completely. So, really, when I got back from Holland, '56, I was trying to earn a living doing lettering and anything I could find.

**FR:** So you're now in the U.S. Who do you visit?

**MC:** Everybody! It was wonderful! This was the late spring, early summer of 1960. I got handed around, you know. I was at Push Pin, and Lubalin, and everyone, and of course I went to Mergenthaler, and kind of fell in love.

**FR:** And who took you around at Mergenthaler?

**MC:** Well, Mike Parker principally. I had met him in England because before he went to work at Mergenthaler, he had had a year's fellowship at the Plantin Moretus Museum in Antwerp, doing a huge amount of work in cataloguing this astonishing collection of 16th century typographic material which survives there miraculously, photographing it, and so on. When he came back, I think in '59, he went to work at Linotype as Jackson Burke's assistant; Jackson was Director of Typographic Development. So Mike could work for him, and when I got there in '60, Mike … I spent a lot of time with Mike and it was he who showed me around.

**FR:** The immensity of that building and all the things that were happening there. The letter-drawing office was gigantic at that time.

**MC:** Yes, yes. Yeah, it really fired my imagination. And they were all very kind to me. I mean, Jackson was incredibly kind to me, and, I mean, people talk rather glibly about culture shock, but boy, did I have a case of that when I got to New York. You know I'd grown up in a sort of rather cozy situation because of my dad's contacts and so on. I knew Stanley Morison, I knew Beatrice Warde, I knew Jan van Krimpen, I knew all these people. I thought I knew everything. I got to New York but I knew nothing, absolutely nothing. I had to start over.

So that was a shock. And I think my first reaction was rather cowardly. I thought, I'll just go back home with my tail between my legs and pretend that none of this happened. But I really couldn't do that. But I sort of let Jackson and Mike know that I would love to work there, but actually it was a blessing in disguise. There wasn't a place for me there then, and I think that was a very good thing because I really had nothing to offer at that point, but five years later I had done some things in London that did sort of equip me for that very form of experience.

**FR:** And how did they approach you to retain you?

**MC:** At Mergenthaler?

**FR:** Yep.

**MC:** Well, I'd kept in touch with Mike, and Mike would periodically come to London on Linotype business, and he would come and stay with me in my flat and the Linotype people were horrified because they thought he was consorting with the enemy, which he was, because I was working for Photon/Lumitype, who were Linotype's competition in the photocomposing world. But Mike and I — those of you who knew Mike, he liked to talk — and we had a lot of discussions in those visits to London of his, about the possibility of my going to Mergenthaler, and indeed, what I would do when I got there.

So when Jackson retired, in I think '63, fairly soon after that the conversations got a little bit more pointed, and we were working towards my actually going there. It wasn't just an idea; that's what we were actively planning. And so in the fall of '65, it happened, and I moved to Brooklyn. I should say, this is occasionally ... I'm having pleasant recollections.

I should say how it is that Mike got the job of being Director of Typographic Development. Here's what happened. You know, Jackson had had that job for a long time. He had to go into hospital, I can't remember exactly why, and he knew he was going to be out for a while to recover, and so on. So of course he told his boss, Jack Keller, that he was going to be out, that sort of thing. But meanwhile, he had privately decided not to come back. He didn't tell his boss that, so of course when Jackson went in hospital, Mike started picking up all the work. I mean anyone who had questions came to Mike. So as the weeks went by, Mike started doing the work. And when Jackson finally did say to Jack, "I'm not coming back. I'm retiring", they said, "Hey, Mike's been doing the job. He might as well have the title and the office".

I think if Jackson had *not* done that sort of subterfuge, they would have thought that Mike was too junior; he'd been there three or four years. They would have gone into the newspaper trade, hired someone, and we never would have had Helvetica.

**FR:** There's a story here. Mike did not have the title that Jackson Burke had. His title was Type Designer. And they paid him less than they paid Jackson Burke. It was very controversial at the time, by the way, because I was on the inside, and ...

**MC:** Yes. When did you get to Mergenthaler?

**FR:** In '59.

**MC:** Aha!

**FR:** I graduated high school in June, and I went to my guidance counselor and I said "I need a job", and he said, "Oh, there's this company Mergenthaler." I said "What do they do?" and he said "Something to do with books." I said, "Sounds interesting." It was in the shipping department. But then I worked my way up through mail boy, clerk, ...

**MC:** You were mail boy when I arrived, there, only a couple of months after I arrived, when I ...

**FR:** That's correct. So you came on my radar because you had a cubicle on the eighth floor, I think.

**MC:** Yes! This was a really, really nice office! It was a corner office, and it was carved out of the steno pool and the filing department, beyond which was the order department and the mail room, by the way. And it was a very nice office. Mike and I found some old furniture in a warehouse that we used in my office.

And it was there, only a couple of months after I arrived, when I got there, I worked all hours, every weekend. I stayed with the Parkers and I didn't find an apartment for weeks and weeks and weeks and weeks because I didn't have a single day off to look for an apartment. We just worked the whole time. So I was working one evening about this time of year — I think it was the 9th of November — when the lights went out. I thought oh crap, the fuse is blown or something, so I looked around and the building seemed to be kind of dark, and I looked out the window and the Navy Yard down below was dark, and I looked over to Manhattan midtown ... dark. And I thought, hello, this is not a fuse. [laughter] This was the famous blackout of the whole Northeast. It went down as far as Maryland, I think, and Ontario in Canada, caused by what was called cascading of the electrical grid. You know, one unit failed; that overloaded the next one, and that failed, and ... cascade.

The typeface I was working on when the lights went out we didn't have a name for at the time, so it's called Cascade. [laughter] And if you can find ... naming typefaces is a lot harder than designing them, so if one falls into your lap like that, you're very grateful.

**FR:** Well, you can copyright the name, you can't protect the typeface. That's the problem in America.

**MC:** That's true. Don't get me started on that. [laughter]

**FR:** So you came on my radar when I noticed that people were going into your office with these big sheets. [holds up a large sheet of paper on which is the drawing of a letter]

**MC:** Yes.

**FR:** I had no idea what this is. I knew these were from the London drawing office ...

**MC:** Yes.

**FR:** And so what did you do? What were they asking you to do?

**MC:** I never made these drawings. These are what are called ...

**FR:** You didn't make them, but they asked you questions about them.

**MC:** God knows.

**FR:** Really! Okay, that's interesting.

**MC:** No, these were produced by the letter-drawing department at Mergenthaler, and had been in that form, not exactly from the foundation of the company, because we once found some of the original forms . . .

**FR:** We have all the original ones here, by the way, and they're very similar, but not the same.

**MC:** Yeah. These, they're like engineering blueprints, they're dimensioned, and so on. In the intervening time between my visit to New York in 1960 and my going down to work in '65, part of the time I had spent working at Crosfield Electronics in London, who were the manufacturing agents in Britain for the Photon machine, Lumitype so-called in Europe. They made the machines, but all the fonts were made in Paris, by Deberny & Peignot, which meant, happily for me, I spent about a week out of every month in Paris, and got to meet and know well Adrian Frutiger and all the other designers in the office. And they were not drawing like this. They were drawing positive. I mean, these are all wrong-reading, as required in the factory. But at Deberny & Peignot, we drew right-reading, and we drew a reasonable scale, maybe caps were four or five inches high. And originally we used scratchboard, but that's not a very good medium; it's not very stable. So eventually we discovered a mylar drafting film.

So when I, in these preliminary conversations I had with Mike, I said I'm not gonna make the things, because I've become very familiar with . . . So Mike explored in the grid-making department where they had a number of cameras, specially built cameras. They were very beautiful things. They were built on a granite slab, to be dimensionally stable. And they found one that had been made for a certain purpose, but it was no longer in use. But it was too big to get rid of, so it was sitting there in a corner.

So Mike figured out what the ratio was between the object and the end of the proper lens, and so we did a little math and we came up with a drawing scale, so we could use this camera which was standing idle to make the plaques. (I don't know if you've got a plaque there, perhaps not. This was a sort of intermediate thing in making Linofilm grids.) So I was able to continue to draw black ink on mylar drafting film.

From the minute I got to Brooklyn to Mergenthaler, to my huge relief, because I didn't want to work in this way [indicates the paper still being held by FR], and also the drawing office was, of course, a union shop, and the letter drawers were only allowed to do two drawings a day, whether they were double-f ligatures or whether one was a period and one was a colon. So that was the day's work. So that was rather limiting in a way. So I walked into a very agreeable, amenable, congenial situation at Ryerson Street, which in a way had been constructed in my honor, because, as I say, no one had worked in that way before. But it was quicker, and more direct, for photocomposition.

LINOTYPE COMPANY LINOTYPE COMPANY
LINOTYPE COMPANY
MERGENTHALER
LINOTYPE
COMPANY.
Brooklyn Factory, Erected and Owned by the Mergenthaler Linotype Company
12 ACRES OF FLOOR SPACE; 2,500 EMPLOYEES
This big industrial plant is behind the Linotype; and behind the plant is an organization constantly planning and working to make Linotypes more efficient and serviceable, that Linotype users may obtain the biggest net results from their investments.
FACTORY OF 30 YEARS AGO: ONE-HALF ACRE OF FLOOR SPACE, 200 EMPLOYEES
Mergenthaler factory building at 29 Ryerson Street, Brooklyn (before 1920). Additional buildings were constructed later. `archive.org/details/linotype-factory-brooklyn-pre-1920`

**FR:** So, from these drawings, they would trace them in a pantograph, and they would create a pattern plate. And by the way, we found three pattern plates in the black boxes, and they were M O P, which is interesting. Then, when they did this, they would trace it in another pantograph and they would produce the punch. And that's what made the matrices which ran in the Linotype machine.

**FR:** It was . . . the number of people was gigantic.

**MC:** Yes.

**FR:** There was a whole floor of people, then there was another floor where people did ancillary work. The whole company was really revolved around these, because this is where they made their money.

**MC:** Yes, absolutely.

**FR:** We processed a hundred thousand of these [holds up a matrix] every day, and they sold for an average of 31 cents apiece, and they wore out. I fact, Ottmar [Mergenthaler] once wrote to the company and said, "I can make them so they don't wear out", and they said "Nah, never mind." [laughter]

**MC:** Yeah, the punch presses shook the building!

**FR:** Oh, yeah, I was just going to say, when they were casting, you *knew* they were casting. They were punching out stuff.

So [picks up the letter drawing] we have all of these drawings, by the way, in the type vault in the back. They went to the Smithsonian, and the Smithsonian didn't want them, and so we [the Museum of Printing] wound up with them, and they are, this is the history of type. Now, no designer of type ever did these. You gave drawings to the letter-drawing office, and they produced these drawings. And the notes on these, some of them are really interesting. There'll be little notes that say "per Mr. Griffith" ...

**MC:** Yes.

**FR:** ... or "per Mr. Jackson Burke", and there were other notes on it. When they converted to phototypesetting, they read the notes. This is from 1943, and it was designed by Dorothy Abergard.

**MC:** I don't remember her.

**FR:** I don't know her either.

**MC:** Each of these, these drawings were all kept in a folio box, and at the bottom was a sort of cheat sheet that told you a breakdown of the essential dimensions and any little bits of history, what Mr. Griffith said to do, and so on. So they give you a little capsule history of how this face came into being.

**FR:** We found phenomenal notes in them. By the way, it all started about 1917. Chauncey Griffith was a salesman for Linotype in Kentucky. And he wrote to the company and said, "Our type is terrible." And so they brought him to Brooklyn and said, "You're in charge." He wiped out everything they did before and started from scratch. And he created a world-class typographic library. In fact, if you look at the typefaces we use on a regular basis, they go back to most of the Linotype typefaces of the 20th century.

He was succeeded by Jackson Burke, and he was succeeded by Mike Parker. Mike eventually got the title Director of Typographic Development ...

**MC:** Yes.

**FR:** ... and he had the best office at Linotype.

**MC:** Yes.

**FR:** He had these wonderful built-in bookcases, and he had all the archives of the company. He had Ottmar's notebooks and he had these big white sheets of paper on his desk. I was delivering the mail to him one day, and I had read a memo about Clarendon. And I said, "Mr. Parker, what's a Clarendon?" And he clears his desk and he draws the history of the serif for me. [laughter] He was that kind of a man. By the way, there's a picture over there of Matt and Mike. So, we're in the hot metal era, but now you see that they're getting into phototypesetting.

Matthew Carter and Mike Parker at the Mergenthaler office.

**MC:** Indeed ... I think it was, at the moment I arrived, the letter-drawing department had broken the back of the work of converting the metal library to film. They hadn't finished it, but most of it was done. And so a lot of the things that Mike and I talked about before I got there, and obviously after, were kind of concentrated on, were there styles of type that had never been made for slug machine use, for technical reasons — you couldn't kern, you couldn't so — so you had to duplex them. But you *could* do it for photocomposition. So that was really, I think, what got me hired. There was an opportunity to take advantage, technical advantage, of photocomposition, in certain respects.

**FR:** Now, Matthew mentioned duplexing. This is one of the limitations of the Linotype machine, because every matrix had two typefaces on it. One was the regular and one was the bold or the italic, which meant that the bold or italic had to be modified to match the width of the regular font. When we went to phototypesetting, we no longer had that limitation.

**MC:** Yes. So much of the work that I just mentioned that the letter-drawing office did of adapting the metal library to film was taking the italics off the same widths as the roman and putting them on their own natural widths, which obviously greatly improved the phototypesetting italics.

**FR:** So when they moved to phototypesetting, they had to produce the first fonts for film, and that was the Linofilm machine. [holds up a square grid] And this is a grid from the Linofilm. And by the way, these were pricey. You put them in a little unit that revolved and then picked them up and then put them in the line for photography and then had a way of selecting the character and exposing that through a lens to size it and then exposing photo material, film or paper. To make these, there was an entire room, which Matt mentioned, in the basement. And it was on hydraulic lifts with a granite block, I think. One end was these plaques. By the way, we don't have a plaque; I should find one at some point in time.

**MC:** Yes.

**FR:** And each one was a different letter, and it was high; it was like eight feet high by eight feet wide, and they would put all these plaques in there. And there was one for every letter. And at the other end of the room was a camera where they had film in it, and they would ... No, no, it was a glass that had silver halide coating, and they would expose it. And that's how they created these grids, which made them extremely expensive.

**MC:** They had to have this room because Ryerson Street was very close to the Brooklyn–Queens Expressway, so the vibrations were phenomenal. This room was called the "floating cloud", and the whole room was on the springs.

**FR:** That's right.

**MC:** [takes hold of the grid] Here, let me handle this. There were four of these on a kind of windmill, and they were each brought into the path of the light with a bicycle chain, weren't they?

**FR:** That's correct!

**MC:** Yes. So, these machines, I mean, they were full of thermionic valves and clattering relays and so on, but there was also a mechanical part of them that was very strange.

**FR:** By the way, they were later sued for patent infringement, and paid a million dollars to Photon over some of the patents involved in all of this. Now, other companies that were getting into phototypesetting created artwork in different ways [holds up several examples] and I've got a whole collection here of some of the different ways they created artwork for phototypesetting. And again, they would photograph these in various kinds of cameras. Some of this comes from Intertype, some from Compugraphic. They were all different in that regard.

**MC:** But all more or less the same scale that I was working at. I mean, this is a very handy scale to work. Big enough so you can get the edge quality right, but it's not too big that you can't see what you're doing. The problem with those ten-inch drawings is, it's really hard to visualize what this is going to look like at eight point ...

**FR:** And so they got through the Linofilm, then they created a cheaper machine called the Linofilm Quick, which really bombed; it didn't do very well at all.

**MC:** Yes.

**FR:** But the machine that made Linotype was the VIP. The VIP — Variable Input Phototypesetter (and we have one in the back, by the way) — and this was the font for it. [holds up a smaller, rectangular film grid] By the way, this was the text one. There's another version of this that was bigger for doing display type. And this is where Linotype really excelled. This is where they made their money, if you will.

However, there was another guy who made as much money and his name was Leonard Storch, and he also made these, and Linotype sued him. and they lost! [laughter] So he made a fortune making fake fonts, if you will, but they were cheaper than Linotype's.

But this was a phenomenal marketplace, and this is how I got into publishing. I published a newsletter for VIP users called VIPPY. [laughter] And by the way, Linotype didn't like it, because I could tell things about the machine that no one else would tell you. So they sued me for 13 million dollars. [laughter]

**MC:** I never heard about that.

**FR:** It never went to trial because when we did the discovery phase, they discovered the terrible mistake they made in suing me, and so they settled by giving me a lot of money, $80,000, to go away. So I built an addition on my building and called it the Mergenthaler wing. So in any case, the VIP to me is a very special machine in many ways.

**MC:** It had very significant consequences of a technical kind. You know, designers are not supposed to like engineers. You know, there's supposed to be one of those sides of the brain problems, but I've always liked engineers enormously and liked working with them. And one of the best experiences had to do with the VIP. Without getting technical about this, the Linofilm, the big Linofilm we called it, the big blue Linofilm, you could not have a zero-width character. I could explain why, but I won't.

On the VIP you could. I've never been 100% sure whether the engineers really understood the significance of that typographically. The reason was that the writing prism in the VIP was driven by something called a stepping motor, which was fallout from the space program. You could send pulses of electricity to it and it moved, you know, jerked along, but you could also *not* send a pulse, so it stayed still.

Suddenly you could do Greek with accents, you could do Devanagari, you could do a whole range of non-Latin scripts (as we called them at that time) that were really not possible by previous means. So I had a very nice period of going to Athens a number of times because a very energetic agent in Greece realized that this machine was perfect, he could sell a lot of them, but there were no Greek types. So I did Helvetica Greek, Baskerville Greek, Century Schoolbook Greek, . . . [laughter] Hermann [Zapf] drew Optima Greek at 36 points; he drew everything at 36 points. (I made the production drawings.) So this took several nice trips to Athens to do this. And it was successful. They sold a lot of machines.

So that sort of interaction between the technology and the design is something that has always kind of fascinated me. And I have had several experiences of that, of working with the engineers or telling the engineers things that we wanted them to consider, that would be very advantageous to us typographers and so on. So that's always been kind of contrary to what designers are supposed to . . .

**FR:** Now when they went to the 54-unit system, that allowed you to do much finer spacing . . .

**MC:** Exactly. The big Linofilm was 18 units and the VIP was 54.

**FR:** Yeah. That made a big difference.

**MC:** It did.

**FR:** You also worked on the 505, I assume.

**MC:** Yes.

**FR:** That was a machine invented by Purdy and McIntosh in England. It was a cathode-ray tube. The characters were scanned from a grid and exposed through a cathode-ray tube as sort of an intermediate approach, but it had a problem in the number of fonts, and Mike came up with this idea for slanting the roman to create the italic.

**MC:** I don't know, . . . , by the way, just because we're recording history here, when I was working at Crosfield I visited Purdy and McIntosh, and I saw the very early stages of this machine. And I told Mike about it and he went and looked at it, and next thing you knew they bought the whole company. So I was never given any credit for that. [laughter] But I claim that.

**FR:** And the reason they did that was they had developed a machine with CBS Labs called the Linotron 1010 . . .

**MC:** Yeah.

**FR:** Later they named it that, and they sold several. They sold one, two, to the Government Printing Office, two to Wright-Patterson Air Force Base. One was supposed to go to Ford Motor Company, but they turned it down. And that was it. There were no other machines. It was too expensive, it was too big. Later on, the Government Printing Office got rid of it. I was in charge of the publicity for the machine. We got on a show that Walter Cronkite mentioned the machine. It was great publicity, but they realized it could never be a commercial success, so that's why they needed another machine and that was the 505. So 1010, 505, . . .

**MC:** Yes.

**FR:** The numbers tell you nothing about the machine.

**MC:** Again, it was a hybrid machine. The reading end of it was digital, a CRT, but the input end was scanning, a thing very much like a Linofilm grid. So the laydown speed was phenomenal. But changing fonts was again, some sort of windmill thing that brought another font up into the . . .

**FR:** The next machine that made them was the Linotron 202 . . .

**MC:** Yeah.

**FR:** . . . which Derek Kyte created in England.

**MC:** Yes.

**FR:** And that became a phenomenal success. And that was a pure digital machine. Your fonts came on floppy disks . . .

**MC:** Let me interrupt you, because I haven't finished.

**FR:** I'm sorry!

**MC:** About the 505. Because the font change was so slow, people started making electronic versions of the distortions. It's like, if you get your TV set set up wrong, the raster goes to hell. You *can* do that under control, so if you do a shear distortion of the raster, you get an italic. It's not the italic, it's a slanted roman. But people started to do that just to save the time that it would take from going from Helvetica roman to Helvetica italic with a font change. Or, by going wide, stretch the raster and out it goes, or you condense it, and so on.

So enough of this was going on that Mike came to me, and he said, you know, Helvetica was not designed for this; Futura was not designed for this. Supposing we designed a sans serif where the geometry was specially configured to do some damage control. In other words, we're never going to make it look right; it's never going to be a true italic, but maybe it won't be quite as ugly as slanting. So we did this, and we made a special typeface, a sans serif, we called it Video, which was really damage control, typographic damage control, and it *did* mitigate some of these horrors that came from fooling around with the raster.

But of course, and this is another parable, no sooner had we done that than they came out with the next machine which had an electronic font change. No loss of time. Video died a death; not for the first time in my life designers were asked to solve a problem, an engineering problem. But engineers are smarter than designers in the end, and they fix the problem, the engineering problem, and designers are left with a solution to the nonexistent technical problem. But you could say that it's worth doing things like that because these machines go through shakedown cruises and …

**FR:** Yes. So you and Mike leave about the time of the 202 or right after that?

**MC:** We left in, um, was it '80 or '81? One or the other.

**FR:** Yeah, the president of Linotype was a guy named Smith, who was a complete idiot.

**MC:** Yeah. He came over from the British government.

**FR:** Well, he was an American that somehow ran the British operation.

**MC:** Oh, yeah.

**FR:** No one figured that out at all. And then, after he left Linotype, he started a company up here in New England, which he put out of business very quickly.

**MC:** Oh, really! I didn't know that.

**FR:** Yeah. That's how we wound up with all the font libraries from Photon and all those.

**MC:** Yes, I see.

**FR:** In any case, you and Mike are now free. Had you decided to do Bitstream before or after?

**MC:** Before. Here's what happened. You know, during the '70s, thanks to the VIP, which, by the way, could set much bigger sizes … Linotype had been a text company, text type company. You could only go to 36 point on a Linofilm, but the VIP went up to 72 at least?

**FR:** On some models, but not very many.

**MC:** Okay. Anyway, it opened up the prospect of display typography for Linotype pretty seriously for the first time. So in combination with the British company and the German company, they ran this very energetic, fruitful, type development project during the '70s, which was predicated on the very, very successful sales of the VIP. There was this big population of machines out there, so we could sell a lot of type to them, to the owners of the machines.

But toward the end of the '70s, Linotype's market share started to decline quite noticeably. And Mike and I almost became concerned that we probably wouldn't be able to continue to run as vigorous a development policy program as we had done for several years. So we thought, well, how about we make type its own P&L. Linotype's business traditionally [was] 90% equipment, 10% type. Type was a machine part essentially. Supposing we didn't treat it as that. Supposing we made it its own profit center. But the Linotype management didn't go for that.

Then the other thing that happened that had a very big influence on us was the invention of very high-end, whole-page digital composing systems. Scitex and Camex, principally. There were others. Scitex was an Israeli company originally in the fabric business, weaving business. But they were very smart and they developed these revolutionary machines that went into Time and Newsweek and places like that, and they cost millions.

**FR:** And they emphasized color, which was the key.

**MC:** Yes, and the whole page. I mean, you didn't just set a galley or a line of type, you set the whole goddamn page with illustrations and diagrams, everything. So … but they had no type. They had this amazing technology, breakthrough technology, but no type. So they came to Linotype, then one went to Monotype, everyone, trying to license a library of type. But they were turned down again by the Linotype management who said no, our type is for our machines, and so on. And Mike and I and others really thought this was a big mistake because we thought again, if we made type its own P&L, we could license the type to these companies and we would make a *lot* of money, because we thought they had a very bright future.

So we really wanted to do a Bitstream from within Linotype. We wanted to have a type department *in* Linotype that did its own thing and made good money. But we were stonewalled completely by the management and we felt so convinced about the need for this that we decided to do it, regretfully, I must say, on the outside. And Scitex and Camex would be, we'd been talking to them and Mike and I were very sorry that they were turned down. They kind of gave us a grubstake to get started and a number of designers joined us, and so on, and so we started a company in Cambridge. We weren't technicians. We started in the shadow of MIT because we knew we'd want good programmers, good technical people and so on. We found them. So we did what we wanted to do within Linotype, outside of Linotype, as it turned out.

**FR:** Well, if you're ever looking for the entire digital Bitstream library, I once traded them advertising in TypeWorld magazine for the entire library. I have it upstairs. It's on floppy disks. I don't know how you read them, but that's a different story. [laughter]

**MC:** I have it too.

**FR:** Okay. And Bitstream was a success. You did very well with Bitstream.

**MC:** For a while, yes. I was there for ten years. I only designed one typeface in the course of that time. I mean, my time was well spent, but I was in endless meetings and so on. But the Bitstream strength was the OEM business, licensing type to these big companies. But a decision was made to go into retail. In other words, to go head-to-head with Adobe in the retail market. And I thought this was a perfectly fine idea, but I realized that there was no one at Bitstream who knew anything about the retail business. We were all OEM people and we were pretty good at it. Mike was an OEM person, the head of engineering was an OEM. So the board decided to go into the retail business, and Mike had meanwhile resigned and left. So they hired a president from the OEM business.

We thought, this is crazy, because this is the wrong guy. We need a retail person. So, there were a whole lot of things that combined, really, with Cherie Cone and I decided the time had come. I realized if I was ever going to design any more type, I really had to leave, because one typeface in ten years is not a good batting average.

**FR:** So you left in '91, and in '92 you formed Carter and Cone.

**MC:** That's right.

**FR:** And where were you based?

**MC:** In Cambridge.

**FR:** By the way, Bitstream, when it started, there was no Adobe, there was no competitor, so they were way ahead, and they did very well for a while, but then you get into desktop publishing, the world starts to change. Yes. So now you are an independent company, and you and Cherie, who's a great marketer, are doing great things with typography. What did you learn from all that?

**MC:** From Bitstream?

**FR:** No, from your experience with Carter and Cone.

**MC:** Oh! It's still going on, I'm happy to say. I'm learning every day! We started ...

**FR:** Do you do every typeface under that name or do you do anything as just Matthew Carter? Does it all go through that company?

**MC:** Yes.

**FR:** Okay.

**MC:** Well, we do quite a lot of custom work, so that would not have the Carter and Cone name on it. I mean, if I do a font for the New York Times, it's New York Times. But for the retail library, the Carter and Cone faces are Carter and Cone.

**FR:** Okay.

**MC:** So the time that we started, '91, was a good time, because we were not the first independent type foundry, by any means. Emigre had been out there for a while. Our fellow Bitstream employee David Berlow had started the Font Bureau I think 18 months or so before us. Sumner Stone was starting out about the same time. But there was a realization that there was a third-party market for fonts.

If you had a Linotype phototype or a digital machine, you didn't have to buy your PostScript fonts from Linotype; there were other sources of PostScript fonts, including an increasing number of independent sources. And frankly, most of the interesting development work was coming from independents. And so this was a good ... a moment of birth was quite lucky in the sense that there was a developing interest in independently designed and produced and manufactured type. And when PostScript Type 1 — originally PostScript Type 3 was the only format you could make — but when Type 1 came about, and essentially someone like myself, I could make a font that technically was the same as an Adobe font. It was ... the Type-1 format was fabulous. And Fontographer had seen this coming and they produced a new version of Fontographer that did Type 1 fonts. So there were a whole lot of things that came together, I think partly by luck, that were good for us at that time.

**FR:** You were in the most interesting period in the history of typography, because in '81 we're still in an analog age, [but] getting into some digital. By the time you get to desktop publishing, when you get into the '90s, now Adobe comes in, PostScript starts to dominate in most ways. You had the "font wars", if you will, and during the font wars Adobe won for a while, but then Microsoft competes with them and they settle on some agglomeration of formats. And so now you're into sort of a standardized world of fonts, if you will. You design a font, it could run on any device out there.

**MC:** Yes.

**FR:** So it opens up new markets, and your evolution is interesting, because you worked in the old world of cutting a punch by hand, into the Linotype casting machines, into the old phototypesetting market, into the digital market. Are you not the last person to have done that?

**MC:** [laughs] I don't really know, Frank. I mean, I am very old. [laughter]

**FR:** Welcome to the club! [laughter]

**MC:** Some people have been very kind and sort of congratulated me on this, and I'm happy to take credit for it, but I don't know what else I could have done. I mean, if you're working in the type business, but it goes digital, I guess I could have gone and been a hermit and something, but I've always been interested, as I said, in working with engineers, and with the technical developments, and so I never wanted to drop out, so to say, of the industrial aspect of type, which is really what has interested me. I mean, I would regard myself as an industrial designer, I'm afraid.

**FR:** Really!

**MC:** Oh, yes.

**FR:** What are you working on right now?

**MC:** I have some work from the dear old New York Times, which never seems to stop exactly, but it's mostly new weights and widths, new versions of existing typefaces, and so on. So it's not the most fascinating work, but I'm so fond of them by now that I'm happy to do it. Jordan[3] and I were talking about this. You know, Adobe had announced that, I think, from the end of this year, PostScript fonts will no longer work in InDesign, and maybe not in the Apple operating system. I'm not really sure. And so a bunch of people who have PostScript fonts have somehow thought, oh, my God! I'm going to have to get some OpenType. But this happened to me with a couple of long-standing clients, including Yale.

Many years ago I was commissioned to design a typeface for Yale University. And most of the conversions to OpenType have already been done. But John Gamble, the printer, woke up the other day and found that there were some sort of subsidiary fonts and things they didn't use all that much, but obviously they would have to be converted. So I'd been working on those, and because the OpenType format is open-ended in terms of character set, I can combine what used to be a number of different fonts.

I did this also for Galliard. When Cherie and I started in '91, the first thing I did was a version of Galliard which we licensed from Linotype. And I think if you bought the PostScript font for just the roman and italic of Galliard, there were actually 11 fonts, because there were old-style figures, there were modern figures, there were fractions, there were extras, there were I don't know what. Now you can roll that all up into a single OpenType font. So I've been doing a good deal of that, and that's interesting, to compile these very big character sets in order to make these faces usable into the future, instead of, you know, dying a death on the 31st of December.

**FR:** Do you have any hobbies?

**MC:** Not really. I generally carry a camera, um, and am quite fond of taking snaps; I mean, I don't take it very seriously, but I wouldn't say it was a hobby, but in this bag of mine there is in fact a camera.

**FR:** We haven't seen you use it yet.

**MC:** No. I haven't used it yet. But no, I don't really have, you know I don't play golf or so. [laughter] I don't have hobbies of that kind.

**FR:** Do you work every day?

**MC:** I do *some* work every day. I have to admit that my stamina isn't what it was when I landed at Ryerson Street and worked long hours every day and weekends and so on. I don't work as much as that, but yeah, I'm still working.

**FR:** Yeah. What was the last book you bought about type?

**MC:** I don't buy many books about type. I get given books about type, [laughter] which is very nice. You know probably better than I do.

**FR:** I thought maybe you read something about a book and they didn't give it to you and you said, "Oh, I have to have that."

**MC:** That does happen.

**FR:** I remember your library was sparse, but interesting.

[3] Jordan Goffin, head of Special Collections at the Providence Public Library.

**MC:** I know. Again, Jordan and I were talking about this. People think I will be a big collector of type specimens and so on. I'm absolutely not. I have about three. The reason is libraries. I grew up in London. Bus ride from St Bride Library. Why would *I*, careless starter out, go and buy a Caslon specimen, money I didn't have, rather than go to St Bride's where they've got *thirty* Caslon specimens, some of them probably unique? So I had no need of having my own library as I was first starting the work. I went to libraries, learned to use libraries, which is a skill, by the way.

**FR:** Any children in your family taking up the . . .

**MC:** I have a stepson and a son; neither of them have the *slightest* interest in what I do. Never have had. And when I was showing signs of being interested in all of this, my dad wanted me to do something else, because he said conversation at the dinner table would be more interesting [laughter] if I did something else. I think he was horrified, and thought of typographic discussions at home at the dinner table, which never happened, by the way. He needn't have worried. But no, my offspring have no interest at all in type and typography. I mean, they're computer literate, but not . . .

**FR:** So you work at a computer, I assume it's Fontographer?

**MC:** Yes. Gerard Unger, my dear friend, used to say that there were only two people left that were using Fontographer, but Gerard's dead; I'm probably the only one. I mean, I do have some other tools, particularly for generating fonts, but I'm so used to drawing in Fontographer. I mean, I can do it without sort of conscious thought, so I *do* still use Fontographer. I have to have an old Mac to run it, because it's not been supported, so two Macs, side by side. I draw on one and I do everything else on the other.

**FR:** That's interesting.

**MC:** Works. An extra airdrop. [laughter]

**FR:** And I have to ask you this question and I know it's ridiculous. What's your favorite font?

**MC:** You know, I once read an interview with Margaret Atwood, the novelist. She was asked the same question about her books, and said, "I can't say, because the other ones are listening." [laughter] So I don't have a favorite of my own or anyone else's typefaces.

What I *do* have is favorite uses of type, but they can change by the day. I can see a book jacket or something or other using a typeface, and I can say, that really makes that typeface look good, whatever the typeface is, or I can pick up a newspaper and see something, and think, that's a really good use of that typeface. So I react very much more to typefaces in play, in use, that I do in any kind of objective way. And where any attempt to, I mean, what's a good example? I have to say . . . Souvenir is not one of my favorite typefaces, I suppose. I really was fond of Ed Benguiat, so if I see Souvenir, I see Ed. You know? So trying to be objective about typefaces is very hard for me, because there are so many associations with people when I see them. I've given up trying to be objective about it.

I'm sorry not to have an answer for you.

**FR:** No, no.

**MC:** But that is an honest . . . I'm not being coy.

**FR:** When you and I lived through that period of the '60s, '70s, '80s, '90s, there were a few hundred typefaces that we dealt with, the classics: the Caslons, the Baskervilles, et cetera. Every day I get three emails promoting at least a hundred fonts at a time . . .

**MC:** Yes.

**FR:** . . . from Bitstream and other companies. There are now, I calculate, over a million typefaces out there, mostly decorative. When you open up Netflix and look at all the pictures of the movies, they're all different fonts, okay?

**MC:** Yes.

**FR:** And yet we still use only a handful of text faces, the classics, more than anything else. How are we going to deal with a million fonts?

**MC:** Beats me. I mean, I don't really know.

**FR:** I mean, just cataloguing them is an impossible thing.

**MC:** I know. I tell you where this is really a problem in the life of people like me, because occasionally I'm asked to judge type design competitions.

**FR:** Ha ha.

**MC:** Ha ha! [laughter] Usually about four judges are doing this. And none of us can *pretend* that we really know whether an entry in the competition is genuine, in the sense that it's not knocked off from something wrong because we *don't* know all those million typefaces. Nobody does, and so on. So in those terms, it's been a problem.

We had a . . . You know I've been one of the judges for the Morisawa competition, and we had a close call once. I mean, we did select a typeface which before the results got publicly announced, someone put their hand up and said, I don't think this is right, and it wasn't. We'd been fooled. We caught it in time, but the day will come when somebody's going to be wrong about that. So that is an actual problem of having so many typefaces.

You were saying there were so few, and one of the first jobs that Mike sic'ed on me when I arrived in Brooklyn was ... you know there's the fifty books of the year and a competition.

**FR:** Yeah. We always do a press release on the Mergenthaler fonts that we use.

**MC:** That's right! My job was to go over to the AIGA who did this because when you filled out the form for your entry, you had to put what the name of the typeface was. But everyone discovered that a great many of these were wrong. So my job was to go and read all of these entries for the 50 winners and correct the attribution of the typefaces, which you could do in those days, because ...

**FR:** There weren't that many.

**MC:** ... there weren't that many, and I knew ... God forbid I should have to do that now. Oh, I wouldn't know.

**FR:** It's an impossible task, and of course Bitstream is, not Bitstream, Monotype Imaging is now the 800-pound gorilla in terms of typefaces.

**MC:** Yes.

**FR:** They bought out most of the big libraries.

**MC:** They sure did.

**FR:** And there are about a gazillion designers. We had a graphic designer here the other day and he gave a talk about how he designed his own font. And so there are now about 20 programs for designing typefaces.

**MC:** Yes. But interestingly enough, maybe it has to do with a million number. As far as I know, Monotype let go all their designers.

**FR:** There are more lawyers working there than there are other designers. [laughter]

**MC:** I don't think there are any real designers.

**FR:** Yeah, again because there are so many freelancers out there who are willing to give them fonts to sell.

**MC:** Exactly.

**FR:** Matthew, this has been phenomenal. When I proposed this to Matthew, by the way, he said, two nattering octogenarians, just what we need. [laughter]

**MC:** But very, very patient. [laughter]

**FR:** So we got to understand a different side of you, and how you evolved, and we appreciate your support of the Museum, so thank you very much. [applause]

**MC:** May I ask a question of the audience?

**FR:** Sure.

**MC:** You know I designed this typeface for Yale University several years ago, and it's been used a lot around the University. And John Gamble, the printer, he sent me an email just the other day and he said, "There's a great revival of interest around Yale in letterpress printing. Students, faculty, alums, they really are into letterpress. Is there any way of making actual type" — I mean, what my dad said, type is something you can pick up and hold in your hand — "of the Yale typeface?"

Does anyone know of any current method, using 3D manufacturing or something, of making actual type?

**FR:** Yes, Ed?

**Ed:**[4] I have a Benton ATF engraver, and I make new faces in hot metal all the time.

**MC:** Thank you. This is worth the trip. [laughter] May I have your card? [laughter] Thank you.

**FR:** By the way, you may have one of the last Benton engravers on earth. I think Patrick Goossens in Antwerp has one, and I don't know of any others.

**Ed:** Yeah. Greg Walters has one.

**FR:** But Greg died.

**Ed:** Yes, but they're forming a not-for-profit.

**FR:** Oh, really.

**Ed:** And it's going to stay there, and David McMillan in Wisconsin has another, but they don't have all the ancillary equipment which is ...

**MC:** It's one beautiful machine.

**Ed:** Yeah, isn't it fabulous?

**FR:** Well, that's what made type, when you get right down to it. Without that ... the Bentons gave us a great thing.

**MC:** Well, thank you. That's my question for you all. [laughter]

[4] Ed Rayher, Swamp Press, Northfield, Massachusetts.

# Prehistory of digital fonts

Jacques André

## Abstract

Over the second half of the 20th century, typography moved from physical metal type to the abstractions of digital computing. This revolution did not follow a straight path. We examine here some of the very first attempts to produce printed characters on computers.

In the 1950s, to satisfy the needs of physicists, the first vectorized letters (and numbers, signs, ...) were made on CRT screens and plotters. In the 1960s, the dot matrix concept allowed consideration of characters as surfaces, leading to digital phototypesetting. In the 1970s, thanks to research in computer-aided design, the way was opened to the fundamentals of digital letter outlines. The first font formats occurred in the late 1970s. The innovation of laser printers, around 1985, marked the beginning of the mature rendering of digital fonts, and the beginning of the commercial font wars, where we will leave off.

## 1 Introduction

Some people think that digital outline fonts were invented by Adobe, others say that they occurred first with phototypesetting, while still others say ...

One reason for this misunderstanding of history is that there have been no detailed and technical historical overviews of this subject[1] (we hope this paper could be a first attempt).

Another, more important, reason is that this story did not follow a straight path, but rather formed a set of rays converging towards the same outcome. At the beginning of the 20th century, book printing was done by experts, both for commercial presses and for institutional documents. In parallel, handwriting became less and less used, while the typewriter industry grew.

During the second world war, a need for a new kind of writing arose: Scientists needed to manipulate drawings and annotate them with letters on brand-new media, such as radar screens. So it was engineers who drew letters as if they were mathematical figures (Bézier, De Casteljau and Karow, for example, were mathematicians or physicists working in the industrial field, and were pioneers in this area, as we'll see). Some scientific developers contacted prominent typographers, for example Higonnet and Moyroud (at Lumitype) worked with Frutiger, while Karow and Knuth worked closely with Zapf. These first research concepts won over the manufacturers, who thus created a new, popular mass market for fonts. I personally think that the success of digital fonts comes from this intimate collaboration of artists and scientists, although it hasn't always been easy!

In this article, we will try to show, without claiming to be exhaustive, many various inventions, even if some turned out to be dead ends. But let us be clear, we do not tell the story of digital typeface designs (even if we happen to cite them), but rather a history of the technological inventions of digital fonts, and the tools for manipulating them by computer. Along the same lines, let's say that this is a story of digital fonts, not of text processing (even if TeX users know that both are related, like METAFONT and TeX).

For lack of space, and also to avoid making this a story of computer science, we have forbidden ourselves to go into many technical details. They can be found notably in the books by Haralambous [57] and Rubinstein [111].

Figure 1 shows the main tools or concepts studied here; it also shows the complexity of this story. We will therefore follow a chronological approach, with interludes to bring together some comparable developments.

## 2 First computerized characters: Line segments

Long before computer data processing, office operations were performed with equipment such as tabulators and printers that used impact technology as typewriters do. Since 1930 two companies were leaders in this area: IBM and (in Europe) Bull. By

Translated and published with permission of the author and publisher. Translation by Patrick Bideault, with assistance from the author and Charles Bigelow.

Editor's note: Preparing this original book chapter for publication required extensive efforts. We profoundly thank the author for undertaking the project at all, after his writing of the original monumental volumes in French [5, 6] (for more on this series of books, see `https://tug.org/books/#andre`), and Patrick Bideault for the translation into English. We also thank Charles Bigelow for the initial suggestion, and his invaluable advice and assistance along the way. Christina Thiele made useful initial translations to get the project off the ground. Thanks, everyone.

[1] In addition to the many specific studies which we will cite below, let us mention some papers such as Knuth's TeX history [79], a master's thesis by C. Knoth [77], and the historical introductions of books on digital fonts like Haralambous's [57] and Southall's [117]. Two important studies (although on a less general topic) have appeared since the French version of this paper was published: Romano's study of desktop publishing [110] and Bigelow's study on the Font Wars [27].

doi.org/10.47397/tb/44-1/tb136andre-prehistory

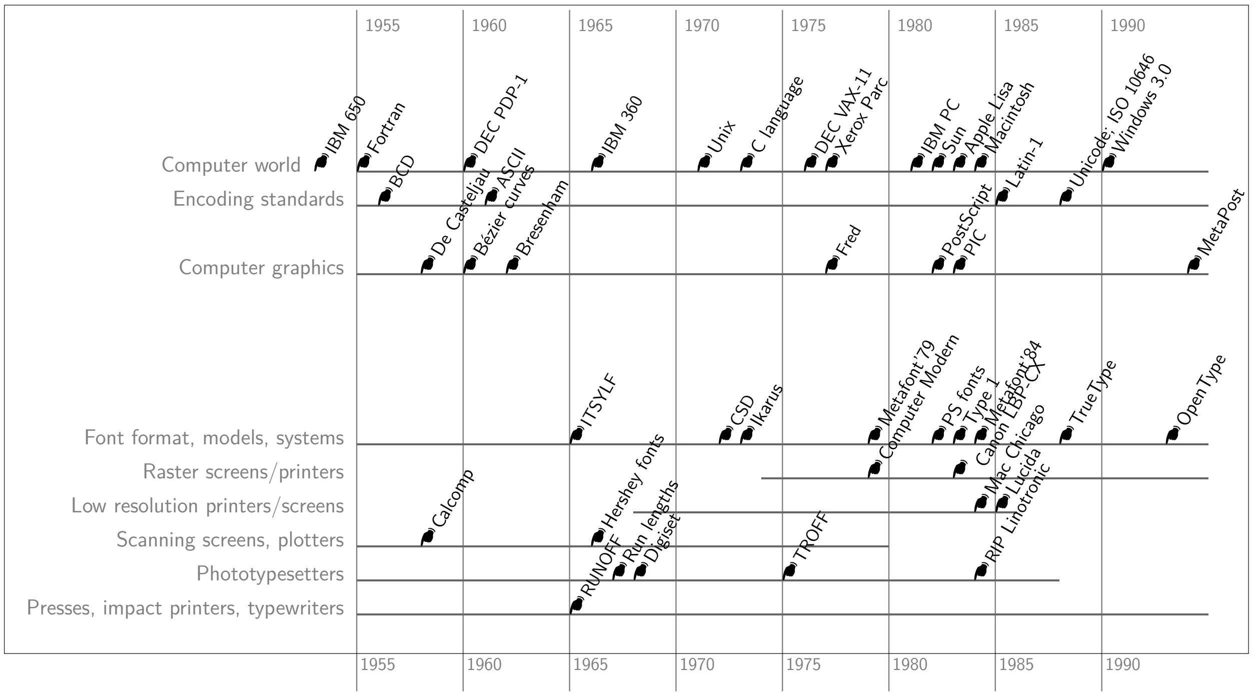
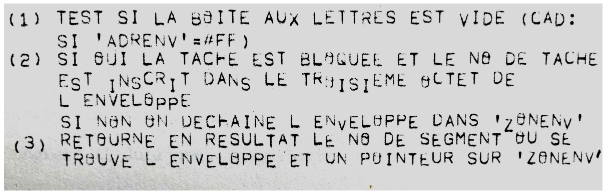


**Figure 1**: Chronology of the concepts and products that led to the birth of digital fonts during the years 1955–1990.



```
(1) TEST SI LA BOITE AUX LETTRES EST VIDE (CAD:
    SI 'ADRENV'=#FF)
(2) SI OUI LA TACHE EST BLOQUEE ET LE NO DE TACHE
    EST INSCRIT DANS LE TROISIEME OCTET DE
    L ENVELOPPE
    SI NON ON DECHAINE L ENVELOPPE DANS 'ZONENV'
(3) RETOURNE EN RESULTAT LE NO DE SEGMENT OU SE
    TROUVE L ENVELOPPE ET UN POINTEUR SUR 'ZONENV'
```

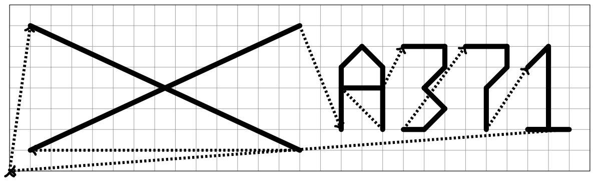
**Figure 2**: Computer output printed by impact devices were not always of high quality... Here, comments in a program [23], 1975.

1945, the first computer outputs were made with such equipment; impact devices remained in use up to around 1995 (a few even later) though today, their printer output may make us smile (figure 2).

Shortly thereafter, around 1950, cathode ray screens and then plotter devices allowed *drawing* of graphics and letters.

## 2.1 CRTs and plotters

Invented near the end of the 19th century, cathode ray tubes saw their first applications (oscilloscope, radar, television) in the first half of the 20th century. But it was not until 1946 that they were equipped with a binary memory that allowed drawings and then alphanumeric symbols to be drawn on them (figure 3). Early displays included EDSAC (1949), the IBM 740 CRT (1954), and others at Manchester University, MIT, and General Electric.



```
0  0  0  % start
0  1  7  % X upper left
1 14  1  % bottom right
0  1  1  % bottom left
1 14  7  % upper right
0 16  2  % A bottom left
...
1 27  2  % 1 bottom right
0  0  0  % return and loop
```

**Figure 3**: Cathode ray screen with XY scanning, and its control program. A spotlight runs along the screen, following the line segment connecting two consecutive points whose coordinates are given. This spot can be lit (thick lines) or switched off (dotted lines). The path, kept in memory, is in a loop which allows the screen to be refreshed (i.e. redisplayed).

It was on these that the first research was done for the basis of what is now called CAD or Computer Aided Design. To control such a screen, it suffices to have a sequence of triplets of the form $(e, x, y)$, where $e$ is 0 or 1, indicating if the spot is lit, and $(x, y)$ the coordinates of the next point. It is these triplets that we will later find in the run lengths of photocomposition.

### Plotters

During the same period, a little after 1950, plotters first appeared, using the same principle of XY plotting as CRT screens. The CalComp 565 plotter, developed in 1958 in California, was the first widely marketed machine and in some ways the archetype of all these products. Other early plotters widely used at that time included the Olivetti XY 600 and the IFELEC 2025 S connected to an IBM 1130 computer.

The CalComp 565 plotter resembles the machine that Nicolas-Jacques Conté had invented in 1800 to engrave the plates of the *Description de l'Égypte* [4, p. 156] (see figure 4) but is electromechanically and computer-controlled. Its operation is analogous to that of the CRT, with the light spot replaced by a pencil that can be lifted or placed on a sheet of paper. This plotter, and all the others, were thus driven by commands sent by the computer according to machine codes specific to each. They operated with only three instructions, quite similar to those of the CRT XY scan (figure 3). To get away from the problem of machine dependency, higher-level languages, such as FORTRAN (notably the PLOT procedure), were soon used.

Plotters were first used in industrial drawing to draw maps for geography, charts for statistics, and so on. These jobs required additional commands, such as "draw a circle with center $(x, y)$ and radius $R$". This was done by using routines that broke the curves into small line segments. In the years 1960–1980 much research took place on the approximation of curves by line segments (curves of degree one), then by curves of degree two, etc. This led to the creation of data-processing languages dedicated to the drawing of curves such as GPCP (*A General Purpose Contouring Program* of CalComp) then HPGL (*Hewlett Package Graphic Language*) which became ancestors of the Fred system at Xerox and from there to PostScript at Adobe (discussed below).

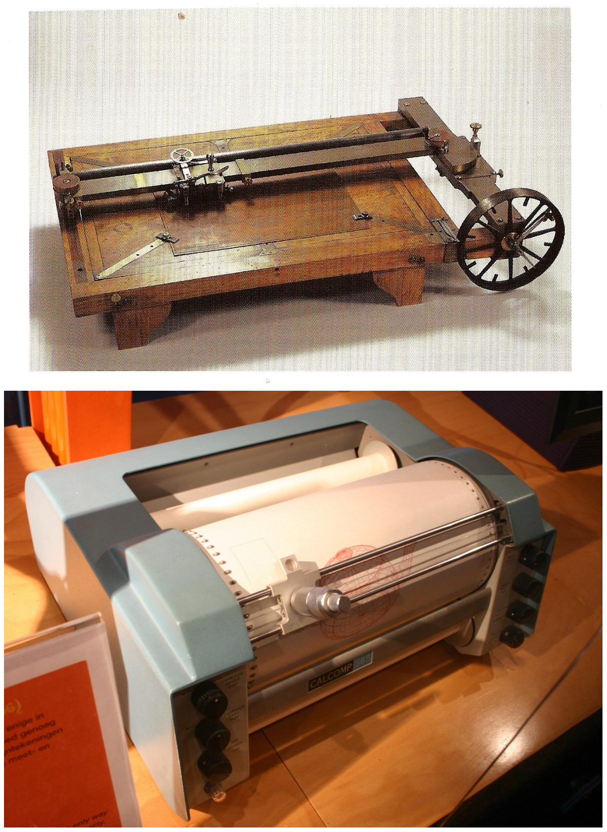


Figure 4: Two drawing machines. Top: Conté's manual etching machine, 1800 [Courtesy CNAM]; bottom: the Calcomp 565, the first electronic drawing machine, 1958 [Courtesy Wikipedia]. (These and following images are grayscaled for print in *TUGboat*.)

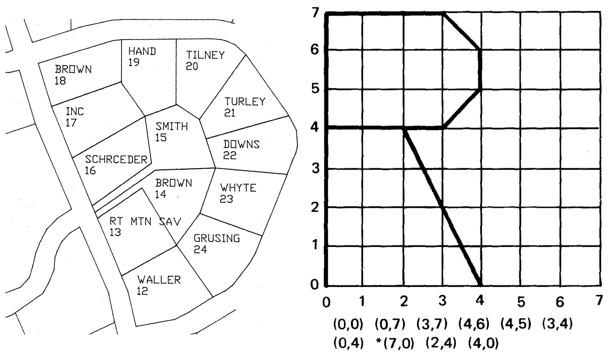


**Figure 5**: Left: extract of cadastral map drawn and written with a CalComp [Courtesy University of Denver Special Collections and Archives]; right: detail of the drawing of a letter R with line segments by a plotter. Extract from a Calcomp manual [41].

### 2.2 Drawing letters with lines

Figure 5 shows that the Calcomp had the ability to draw characters, essential in technical drawing for legends and markings of all kinds. Characters are treated as small drawings formed by a series of line segments (right-hand image). To the characters originally provided in the CalComp 565 (capitals, numerals, and "a few special characters") were gradually added the other characters of various six-bit

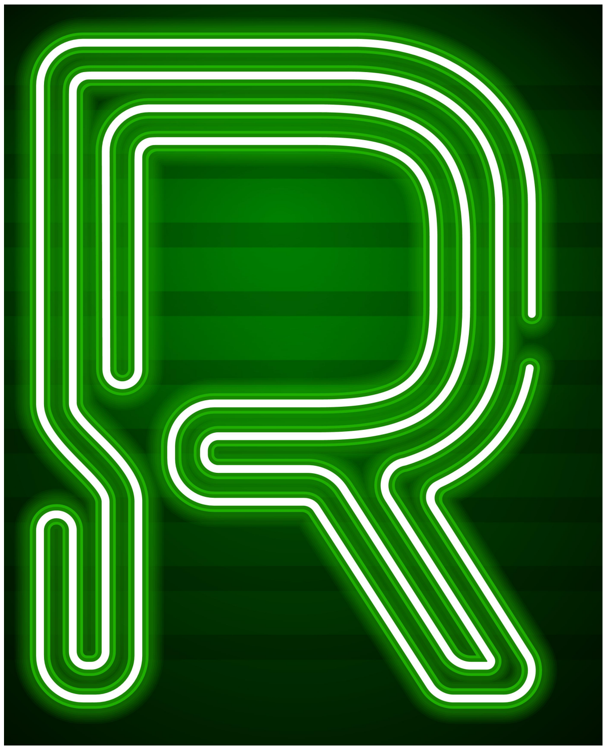


**Figure 6**: This linear neon tube fills the letter R like a Peano curve. [Courtesy Depositphotos]

binary-coded decimal (BCD) codes of the time, and then of seven-bit ASCII, then in its infancy. Eventually, given the extensive use of these symbols, CalComp "hardwired" the symbol plotting instructions, making them very fast.

Other plotters were soon created. One example is the Perthronic plotter from Aristo (Hamburg), which as early as 1960 was plotting numbers using so-called "stick digits", characters drawn with only straight lines. Today, plotters use standard vector fonts.

**Filling characters with strokes.** A figure defined by its outline can be filled in by hand with fairly tight strokes. Foundry catalogs from the 1930s show designs such as Prisma by Rudolf Koch (1931). As early as 1925, Fernando Jacopozzi displayed the letters "Citroën" (the famous French car maker) on the Eiffel Tower by electric bulbs that were aligned on wires (not a pixel array). This technique was used extensively for signs with neon tubes (now in the Las Vegas Neon Museum) and some letters could even be filled in with a single tube using Peano's curves (figure 6). At the end of the 1970s, METAFONT79 offers the concept of "double draw" for filling in between curves [80, chapter 6].

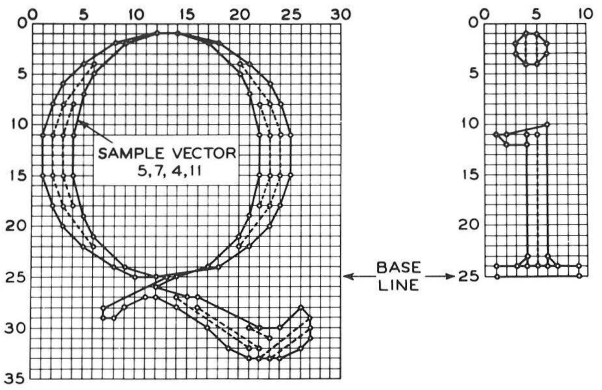


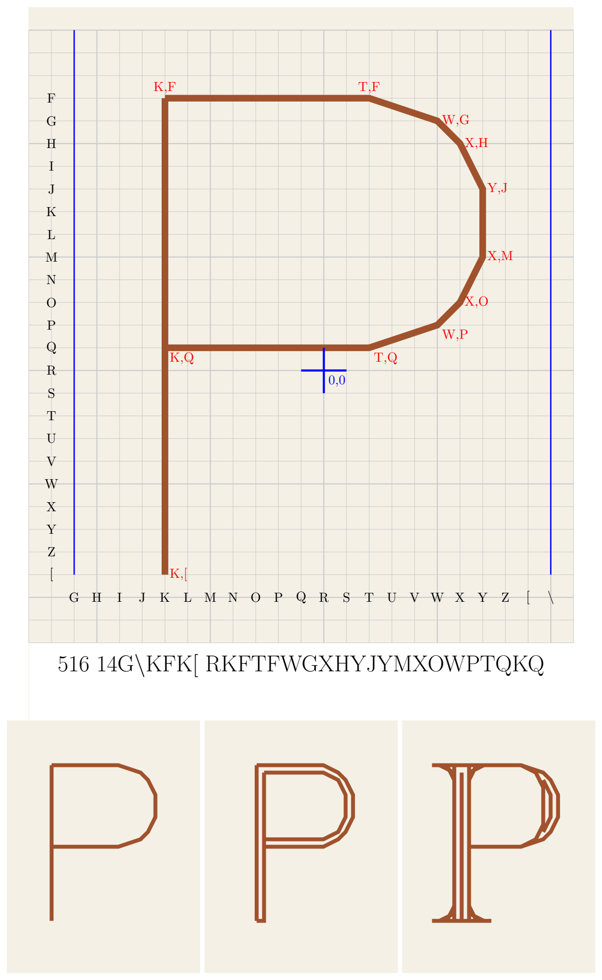
**Figure 7**: First attempts at filling letters with interior tracings: two letters, inspired by Baskerville, from the Bell system, 1967 [98]. [Courtesy Visible Language]

**Bell characters.** Under the direction of M. Mathews, a team at Bell Telephone (Murray Hill, USA) studied, shortly after 1965, a character production system for CRTs producing microfilms [98]. Character outlines were defined using line segments with a keyboard input system that allowed for the definition of several character sizes. Because the plotters' strokes were thin, the letters were blackened by drawing "inner outlines", a technique that would be seen again with Allen Hershey's typefaces. Figure 7 shows the principle.

**Hershey typefaces.** Around 1967, Allen Hershey developed a series of fonts at the Naval Weapons Laboratory (USA) that could initially be used with the Calcomp. Well documented — see [61] and [128] — and virtually copyright-free, they were widely distributed and used in the graphics world for years in their native form; they are still used in vector form today [38].

They were not written directly in the Calcomp language but in their own format, which made it easy to port them to other plotters. For Hershey, a font is a database, whose elements include (in a language called R-code) a glyph number (e.g. 516 for P), the number of points describing the design (14 for P), two "abscissae" to deduce the slopes and the width of the character, and finally the coordinates of the points of the line segments. Each coordinate was given by an alphanumeric sign according to the transliterated ASCII type encoding: $\texttt{G} = -11, \texttt{H} = -10, \ldots, \texttt{R} = 0, \texttt{S} = 1, \ldots, \texttt{[} = 9, \texttt{\textbackslash} = 10$, and so on; recall that at that time available memory was very limited and one had to find tricks to save space.

In general, each character is defined in three modes: simplex (with a single stroke), duplex (two strokes) and triplex (three strokes) simulating three


K,F
T,F
W,G
X,H
Y,J
X,M
X,O
W,P
K,Q
T,Q
0,0
K,[
F G H I J K L M N O P Q R S T U V W X Y Z [
G H I J K L M N O P Q R S T U V W X Y Z [ \

516 14G\KFK[ RKFTFWGXHYJYMXOWPTQKQ

**Figure 8**: Hershey's P pattern. Above: a detailed plot, with the R-code of this "P 516" below; the coordinates are indicated by the letters FGH...
Below: the three Hershey simplex, duplex and triplex P's have different weights, simulated by the presence of one, two or three lines. Drawings created after Hershey's tables [127].

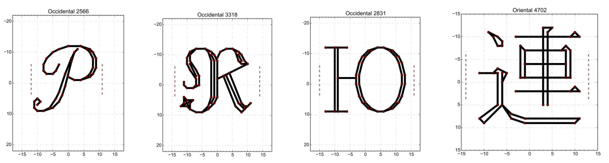
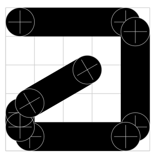


**Figure 9**: Examples of Hershey's characters: round, blackletter, Cyrillic, CJK — all drawn with straight lines. Based on [113]. [Courtesy Stewart Russel]



```
1 setlinewidth
1 setlinecap % rounded ends
0.5 4.5 moveto 4.25 4.5 lineto % top horizontal
4.5 4.25 moveto 4.5 0.75 lineto % right vertical
4.25 0.5 moveto 0.75 0.5 lineto % bottom hor.
0.5 0.75 moveto 0.5 1.5 lineto % left corner
1.75 0.75 moveto 2.75 2.75 lineto % diagonal
stroke
```

christian DELORME

**Figure 10**: Left: an 'a' from Delorme;
right: corresponding PostScript instructions [47];
below: Delorme's name in his font.

different weights. In addition to these weight variations, Hershey programmed a series of style variants (cursive letters, blackletter, etc.), and also non-Latin characters (including mathematical [128] and chemical characters, Cyrillic, and Japanese); see figure 9.

**Microfilms.** The first microfilm systems were equipped with a CRT that also produced text, such as the IBM 228, Alden, Benson, Control Data 280 systems, and others. A special mention to Stromberg-Carlson who, after a 64-character set for XY scanning, offered their 4600 Microfilm Recorder model with 112 characters, also using arcs, thin and thick strokes, thanks to a four-coordinate system in a $4096 \times 3072$ raster [105, p. 172]. These systems clearly influenced the third-generation photocomposers (page 28).

## 2.3 New line-based typefaces

These line-based typefaces have had little influence on digital fonts (except for the microfilm technique), but they have played a vital role in computer science, especially in CAD (Computer Aided Design), and they could not be ignored.

Either for fun, or to simulate old fonts dating back to the first plotters, digital stroked fonts can still be found nowadays, such as VECTOR BATTLE. Others are part of the typographic research of the 1980s.

**The Delorme typeface.** Christian Delorme designed a typeface composed of cardboard strips, rectangular and rounded at the ends, allowing for a much greater weight than that left by the tip of a pencil (figure 10). The connection of the segments of these thick straight lines gave the corners an illusion of roundness. It was digitized in a PostScript font format using the so-called `PaintType=3`, as used for the initial PostScript Courier (discussed below).

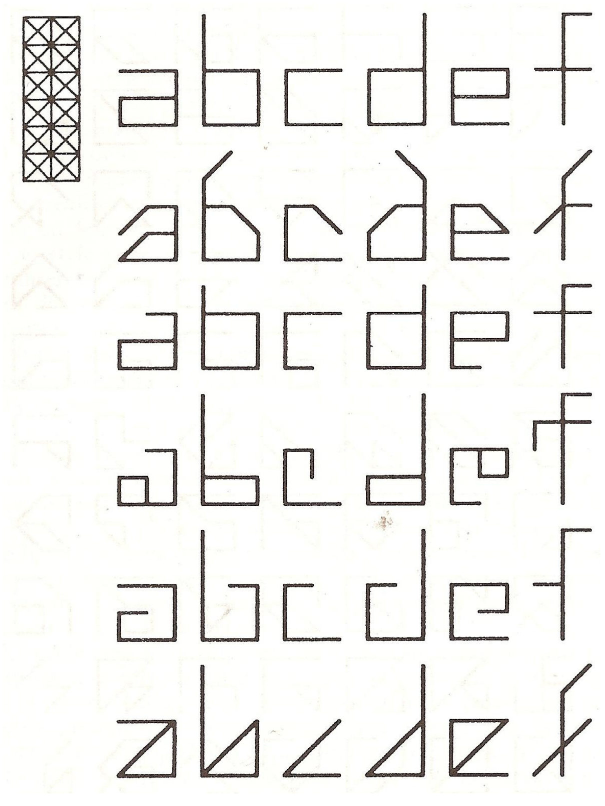


**Figure 11**: These alphabets composed in 1985 only of horizontal, vertical and diagonal lines were designed by computer, each line being letters "in the same spirit". Excerpt from Douglas Hofstadter, *Metamagical Themas* [65, figure 24-14]. [Courtesy Perseus Books]

**Douglas Hofstadter's gridfonts.** Douglas Hofstadter is a professor of cognitive science and computer science, with adjunct appointments in philosophy, comparative literature and other departments, at Indiana University in Bloomington, Indiana, USA. Most famous for his book *Gödel, Escher, Bach: An Eternal Golden Braid*, he is also known for his research on letterforms, including a long essay in response to Knuth's "The concept of a meta-font" [82], collected in his book *Metamagical Themas* [65, ch. 13].

In his work, Hofstadter asks himself the question of how to draw automatically (by computer, using artificial intelligence programs) as many 'a's as possible and then create the rest of the alphabets in such a way that all the letters of a single alphabet (which he calls gridfonts) share "the same spirit" (figure 11). His research is more philosophical (what is "the essence of 'A'-ness"?; what does "in the same spirit" mean?; etc.) than technical. But what we note here is that he uses characters composed only of strokes.

```
%FontType=1 PaintType=3 isFixedPitch=true
40 setlinewidth % bold => 80
0 setlinejoin
1 setlinecap
/A{/base currentlinewidth 2 div def
   120 545 moveto 325 545 lineto    % 1
   520 base lineto                  % 2
   280 545 moveto 80 base lineto    % 3
   30 base moveto 200 base lineto   % 4
   400 base moveto 575 base lineto  % 5
   165 210 moveto 440 210 lineto    % 6
   stroke } def %  A
```

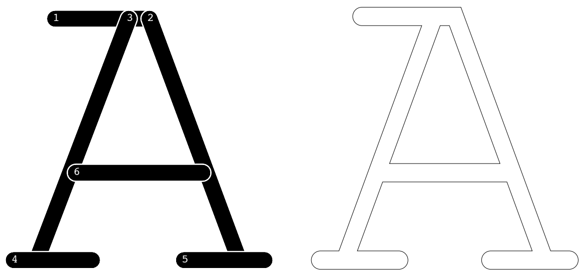


**Figure 12**: Adobe's Courier font in PostScript. Left: building the capital A with six stroked line segments (the PostScript instructions are shown), as used in the initial release of PostScript.
Right: in subsequent PostScript releases, Adobe used outlines for Courier, as with all other bundled fonts. (Excerpts from [12]).

**Adobe's Courier, v1.** To enter the CAD market, Adobe included the then-commonly used stroked fonts in its PostScript language. Fonts supported a so-called `PaintType=3` mode where only stroke instructions were used to draw the character, with the `fill` operation having no effect; the thickness of the strokes could be specified with the `linewidth` parameter.

The first version of Adobe's Courier [12], included in the initial release of PostScript, was defined using only thick strokes with rounded ends. In subsequent releases of PostScript, Courier, like all the other included fonts, was defined using outlines. The two are compared in figure 12, while figure 13 shows a clever use of a fixed thickness to simulate the variable thickness of the apostrophe.

## 3 Initial bitmap concepts

### 3.1 Screens, bitmaps and scanning

The first screens used XY scanning (page 22) but, with the cost of memory decreasing, since 1950 CRT screens with television scanning were in use. TV scanning consists of filling a matrix of points line by

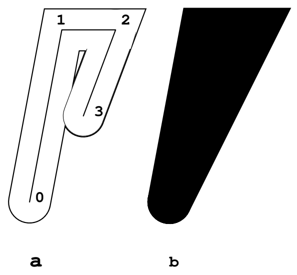


**Figure 13**: Adobe Courier v1 apostrophe construction (from [12]).

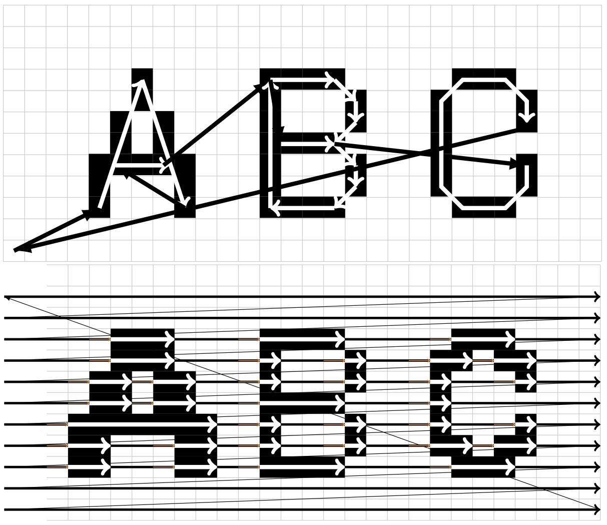


**Figure 14**: Two scanning methods for screens (television, computer, etc.): above, XY scanning (direct, by vectors); below, television scanning. Along white arrows, the beam blackens the pixels of the screen; with thin black lines, the beam writes nothing; raster returns are indicated by a thinner beam. The pixels are enormously magnified as large squares so as to show the scan.

line (figure 14): the usable surface of the screen is scanned from top to bottom, line by line, each one from left to right, with a step as small as possible. Some screens had a different scanning direction; for example, the Digiset scanned vertically (figure 18).

The "carriage return" of the beam to refresh the screen is called the "frame return", the image of the screen being assimilated to a frame.

### 3.2 Frame concept

Canvases existed long before computers, as fabrics appearing in the Western world, as early as the Neolithic period. These fabrics, when they are thick and not too tight, define a kind of grid, and are called canvas. Canvas fabric served as the base for needlepoint embroideries and tapestries: a thread of wool is passed through this grid, thus defining "points" corresponding to the pixels of our bitmaps. Cross-stitch embroidery began in the Middle Ages.

As early as 1600, Le Bé shows models of letters embroidered with a grid of $10 \times 14$ such "pixels". Figure 15 shows that there were already solutions to problems that we will see again with our computer bitmaps: the diagonal of the N is not linear (as in figure 21) and there are white squares at the junctions of the letters; these limitations are used for aesthetic purposes. Around 1750, the Encyclopedia of Diderot and D'Alembert shows very beautiful alphabets on a grid of only $7 \times 7$ pixels [50, Suppl. 3, pl. 4].

In the nineteenth century, with the introduction of compulsory schooling, the embroidered alphabet was substantively developed. The teaching of it was abandoned by 1930.

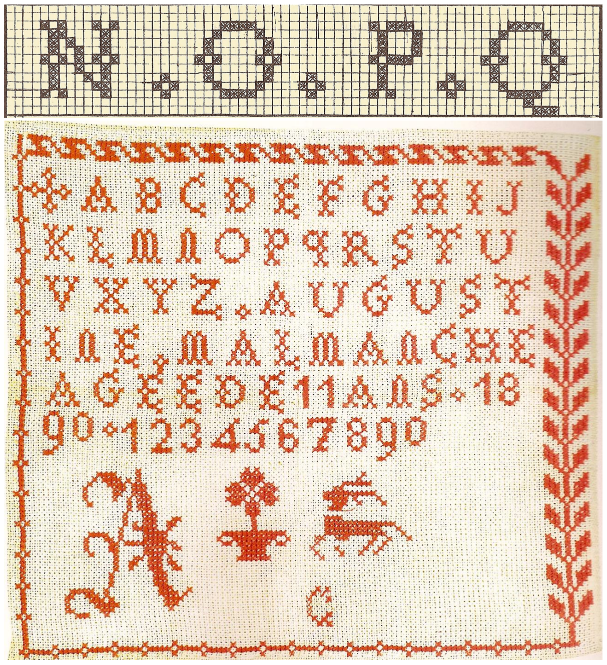


**Figure 15**: Above, excerpt from *Belle Prérie* by Le Bé, 1601 [coll. J.A.]; below, school exercise in cross-stitch embroidery, late 19th century [Credit Stefano Bianchetti/Les Éditions de l'Amateur].

**Woven books.** It is well known that Joseph Marie Jacquard designed at the beginning of the 19th century the first mechanical loom using punch tapes (based on 18th century inventions) and so the first computer automaton (Charles Babbage was inspired by it to make his Analytical Engine [53]). Recent studies [27, 107, 126] pay attention to the fact that this machine was able not only to design graphics but also texts, considering letters as a special case of graphics (as PostScript and METAPOST would

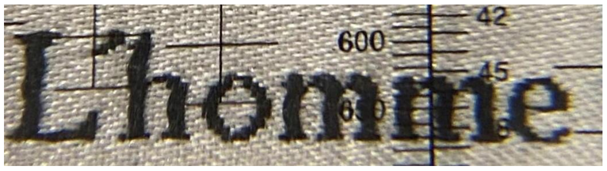


**Figure 16**: Detail of *Les Laboureurs* by Lamartine, Lyons, 1878; shown through a lens, scaled ≈ ×3.5. [Courtesy RIT Cary Graphic Arts Collection]

do a dozen decades later). In the city of Lyons (France), some manufacturers exhibited their skill by weaving books in silk on a Jacquard loom. Among these books, extremely rare today, let us mention *Les Laboureurs* by Lamartine, woven in 1878, and *Le Livre des Prières*, 1886.

The weft thread behaves, when it is over the warp thread, like a black rectangular pixel and when under, like a white pixel. The succession of over/under allows filling characters as in figures 18 and 54 below. The loom mechanism putting thread over or under the frame was governed by punched tapes, according to a bitmap, called "mise en carte". It is a paper with a grid of 1 cm square, each one divided in 10 × 10 pixels. It is not clear exactly how this bitmap was "programmed", to use a modern term. But it is conceivable that the letter images were reproduced from templates or pre-digitized models.

This Lamartine text (figure 16) has been composed in body size close to 8 pt. The jewel-like precision of the book type has a digital resolution comparable to laser printer resolutions of a century later. As Bigelow says [27], these books show the true first ancestors of digitized types.

**Mosaics.** Although the mosaics of the Greek, Roman, early Christian, etc., times often have textual inscriptions (figure 17), they are not true raster letters in the sense that there is no regular raster (neither for the background, nor for the letters). Rather, they are a construction with completely disordered pixels, without being a random raster.

## 3.3 Photocomposers

The photocomposers of the first two generations used characters photographed on film [6, ch. 1, Photocomposition]. Generally, these typesetters were driven by in-house tools. At the beginning of the 1970s, the Unix group at Bell labs got a Graphic Systems CAT phototypesetter. Joe Ossanna then wrote a version of nroff (a text formatter for typewriters or impact printers) that would drive it. A few years later (around 1975) Brian Kernighan adapted troff to C programming, to any kind of second generation typesetter [74], and even to mathematics (eqn language) [76].

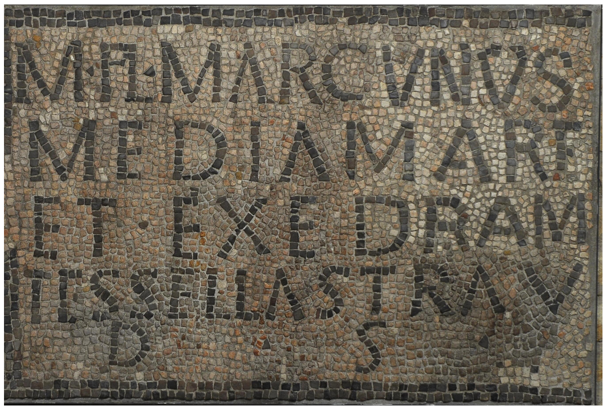


**Figure 17**: Early Christian inscription (CIL XIII 11479) in mosaic tesserae discovered in 1905 in Avenches/Aventicum, Switzerland. [Courtesy AVENTICVM]

The third generation of photocomposers marks the beginning of the use of digitized characters. The first digital photocomposer was created in Kiel (Germany) by Dr. Rudolf Hell [59, 117] whose company specialized in special equipment, photography and electronics.

Hell was inspired by the technique used for the first microfilm systems (page 25). The image of a typeface is projected onto the screen, with a television-type scan (figure 14, but in this case, a small technical difference, the scan is vertical and not horizontal), then exposed, produced from a matrix drawn by a typographer.

To do this, it drew (figure 18) the desired character on a large layer and marked with 1, or X, the boxes to be blackened, the others with 0, or left them empty. A programmer translated this drawing into commands for the (vertical) scanning: number of the column, number of the first pixel to be filled, number of pixels. This is what we call *run lengths*.

## 3.4 About bitmaps

The grid of screens can be considered a matrix with each element being 0 or 1. Each such element is called a *pixel* (abbreviation of *picture element*). If each element is a 0.1 inch square, i.e. if there are 10 pixels in an inch, the *resolution* of this grid is said to be 10 dpi (*dots per inch*) (figure 19). The resolution for phototypesetters was very high (often 1200 dpi, sometimes more), resulting in the naked eye seeing very smooth curves and characters. On the other hand, the screens of the first microcomputers or Minitel (see page 33) had a resolution of only

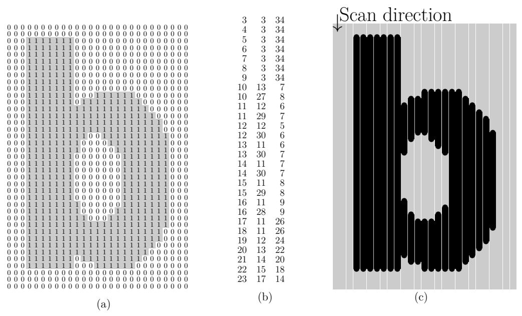


**Figure 18**: Principle of the Digiset: (a) the binary matrix conceived "by hand" by the typographer, (b) code by range (i.e. run-lengths), (c) image provided by the photocomposer: the bands (here slightly narrowed to distinguish them) are scanned from top to bottom (the returns of screen are not indicated); the gray corresponds to a phase of non-illumination (the screen and the paper are not exposed) and the black with a phase of illumination (thus exposed).

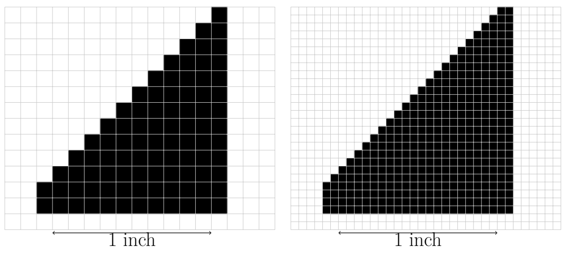


**Figure 19**: The same triangle rendered as a bitmap at 10 dpi and 20 dpi resolutions.

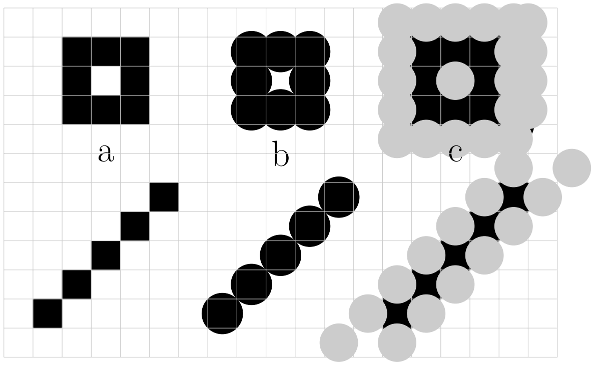


**Figure 20**: Influence of exposure modes: (a) theoretical form; (b) "white then black" mode; (c) "black then white" mode (here white is gray). After Pierre MacKay [92].

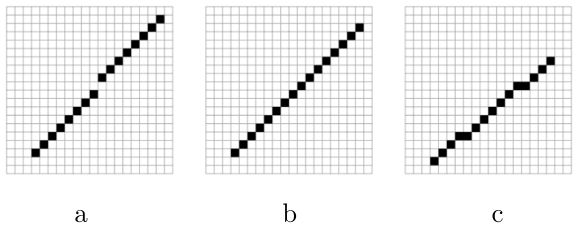


**Figure 21**: Bresenham's algorithm (1962). (a) A line segment drawn directly using the Cartesian equation $y{=}ax{+}b$ with integers; (b) the same line segment drawn using Bresenham's algorithm: a slight shift allows continuity (no breaks as in a); c) another line segment (with less slant) drawn also using Bresenham's algorithm, still ensuring continuity.

72 dpi, with resulting "pixelated" mosaic-appearing characters.

Such a bitmap is a virtual image to be displayed on screen or printed on paper, resulting in a few differences from the theoretical matrix: the pixels which should be square are often round, like the trace left by rays of light (figure 20); in addition, some output devices (in particular, the LN printers from Digital Equipment Corporation (DEC), the Ricoh printers and some from Xerox, still in use at the end of the 1980s, and some of the black and white screens of the time) did not work by blackening a white zone, but by blackening initially all the paper and by then writing, by sweeping, the white where it is necessary. This gave appreciably different results according to the machine used (figure 20).

The underlying problem with bitmaps is that we go from a continuous world to a discontinuous one. One result in particular is that any slant in relation to the direction of the pixels presents pixelations or so-called staircase effects. Several methods have been used to reduce these effects; they cannot be eliminated (even when using vector fonts, contrary to what we sometimes read in the press). First, and most simply, increase the resolution, i.e. decrease the size of the pixels (figure 19).

Second, use a concept of bitmap not based solely on black and white, but with more subtle possibilities, such as grayscale screens (figure 22) that will appear around 1980, and LCD (Liquid Crystal Displays) at the end of the 1990s (page 52).

Third, and most generally, researchers found ways to reduce the artifacts in bitmaps by using techniques from computer graphics. If we draw a line $y = ax + b$ by writing a loop giving to $x$ the integer values $3, 4, \ldots, 18$ for which we calculate the corresponding integer value $y$ and then blacken the box $(x, y)$, we obtain figure 21a, which is not satisfactory since the slanted line segment is cut in two. In 1962, an IBM engineer, Jack Elton Bresenham, who was working on the first Calcomp plotter using bitmaps, looked at the problem. The Cartesian method doesn't work because the rounding done to take the

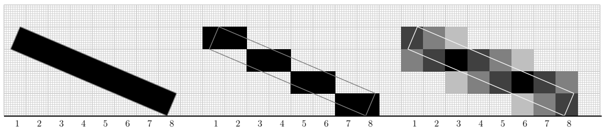


**Figure 22**: A line segment, left: expected; middle: with Bresenham algorithm; right: with Bresenham algorithm on a grayscale screen at the same resolution.

integer part causes such dropouts. This is correctable, but the correction method known used operations on real numbers, which were very time consuming (especially for computers of that time). Bresenham's algorithm starts from the parametric form of the equation of the line and for each point studies its neighbors [39]. It thus manages to optimize the plot by using only integer operations, which is extremely fast; figure 21b shows the result obtained. This algorithm has been improved to deal with borderline cases and adapted to other curves (including circles) and even to grayscale screens (figure 22). This is, in a way, the archetype of all computer graphics programs used in typography!

## 3.5 Bitmap fonts

To these technical problems, type designers brought an artistic solution: circumvent the problem by using no or few diagonal lines.

Thus, Adrian Frutiger, who experienced "the passage from lead to CRT, so greedy in memory, then to vectorized representations [...] then to Bézier curves" [102, p. 286], says about his Méridien font, which had already been adapted from lead to Lumitype, "When I saw what the digitization gave, with all these small stairs, I was horrified. [...] So I tried to get around the technical deficiencies by drawing. It was necessary to avoid the slight curvatures of the slightly curved solids and the concave serifs, which would have made the pixelation visible ..." He then drew the Breughel font, which takes these adaptations into account (figure 23). His specific recommendations were to avoid the stairstep rendering by the absence of oblique lines, in particular, to flatten the serifs; to increase the curvature of the curved solids, or on the contrary to flatten them (left side and right side of the two stems of the 'n'); to prevent the ink traps of the holes by enlarging them, and so on.

Other type designers for the CRT had the same problem, for one, Ladislas Mandel with his Galfra design (figure 24). Hermann Zapf also studied the digitization of a subtle typeface design he had designed for lead, Optima, and refrained from completing this work for printers with less than 300 dpi [129, p. 103] (figure 25).

Despite these shortcomings, and the heavy workload involved, many fonts were designed character by character for the three generations of phototypesetters (figure 26). Many fonts were marketed for or sold with photocomposers.

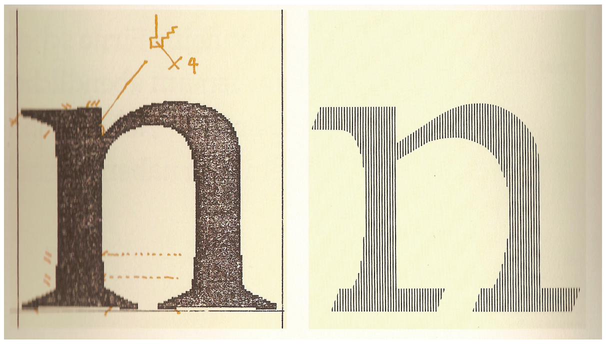


**Figure 23**: The defects of the Frutiger Meridian scan (left) were corrected by hand, resulting in the Breughel design (right) [102, p. 290].

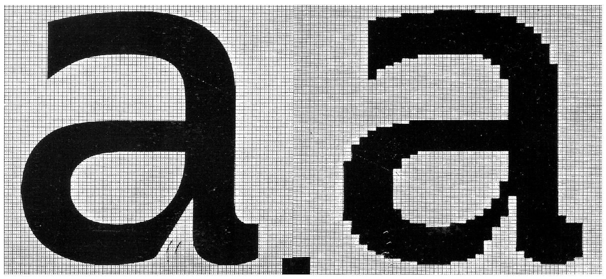


**Figure 24**: The 'a' in Ladislas Mandel's Galfra typeface (~1978): left, hand-drawn; right, pre-digitized. [94]

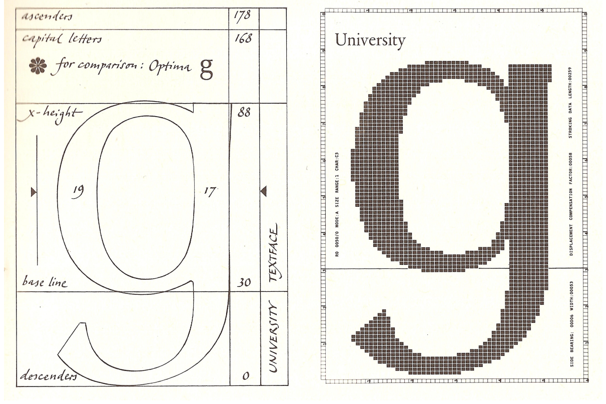


**Figure 25**: Hermann Zapf preferred not to finish this draft of his Optima typeface for a printer at less than 300 dpi, which could not render the design well [129].

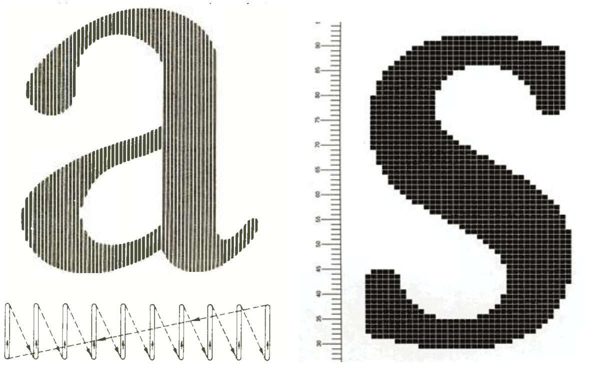


**Figure 26**: Left: ‘a’ of the Videocomp composer (1967) with scan and frame return diagram, from a newspaper, 1971. Right: one of the first fonts for Hell’s Digiset, Demos by Gerard Unger, 1975 [Courtesy Gerard Unger].

## 3.6 Research of new typographies and imitations of pixelated characters

While designers like Frutiger and Mandel sought to free themselves from technical contingencies, others used them as a means of expression.

**Crouwel’s New Alphabet.** The Dutchman Wim Crouwel [18, 45] was attracted to digital fonts, which he discovered thanks to Rudolf Hell. One of the first fonts he designed, directly in bitmaps for CRTs, is New Alphabet, in 1967 (figure 27). Since he could not solve the problem of curves or even diagonals, which are always stairstepped in digital rendering, he decided not to use them. So he adapted his fonts to the machine, using only horizontals and verticals, with slightly diagonalized junctions. But then some characters became unconventional (the A for example!). His typeface was not without criticism at the time. In 1983, Charles Bigelow wrote: “A letter is something other than a collection of bits. All curves are eliminated. All shapes are simplified. From a purely technical point of view, the result is an undeniable success: each letter reproduces itself impeccably, even through the largest frames. This new alphabet has only one disadvantage: it is unreadable. It is unacceptable. Without legibility, there is no communication [...] Who can distinguish letters from numbers?” [24]. We also owe to Crouwel the somewhat more customary typeface Claes Oldenburgh (figure 28).

In addition, the look of bitmapped mosaic characters inspired graphic designers. In 1982, Michel Olyff took up the concept of bitmap by drawing pixels, by hand, one by one for his famous poster (figure 29), inspired by a doily embroidered during the first world war.

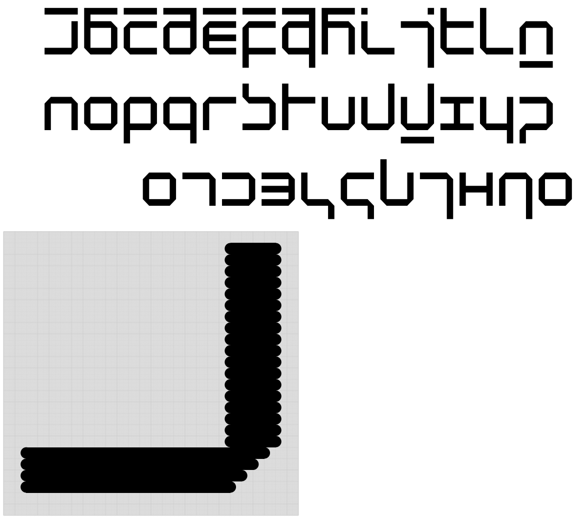


**Figure 27**: New Alphabet is a typeface designed by Wim Crouwel in 1967, banning all curves and diagonals; there are no upper/lowercase distinctions. Above, the digitized version [courtesy The Foundry]. Below, the original ‘A’ (reprogrammed from [18]).

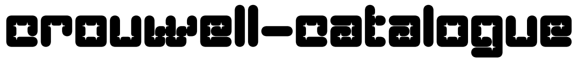


**Figure 28**: The Foundry’s Architype Catalogue font, digitized in 2003 from the Claes Oldenburg font designed in 1970 by Wim Crouwel, which was clearly inspired by bitmap drawings like those in figure 20. [Courtesy The Foundry]

For its part, the Honeywell-Bull computer company used very pixelated characters for its poster of the SICOB computer show in 1982 (figure 30).

## 3.7 Matrix printers

The principle of drawing characters by a matrix of dots has existed since the end of the 19th century; e.g. the *pin points* of punches, characters for Smith typewriters in 1910, and then characters written at the top of IBM-026 punched cards. It was taken up by *dot matrix* printers in Japan in 1968 and spread to the USA via the LA30 and then LA36 from DEC.

The basic principle is as follows: a print head comprises a number of vertically-aligned pins allowing the spontaneous generation of a column of points. They are propelled by electromagnets and, through a carbon ribbon, print the points of the character images. The characters are defined by a matrix of points, often $5 \times 7$ but which, by shifting, in fact defined 9 lines — figure 31 shows the principle.

A $5 \times 7$ matrix does not give good results, so manufacturers improved the quality of the characters, while often providing two modes: one of draft quality,

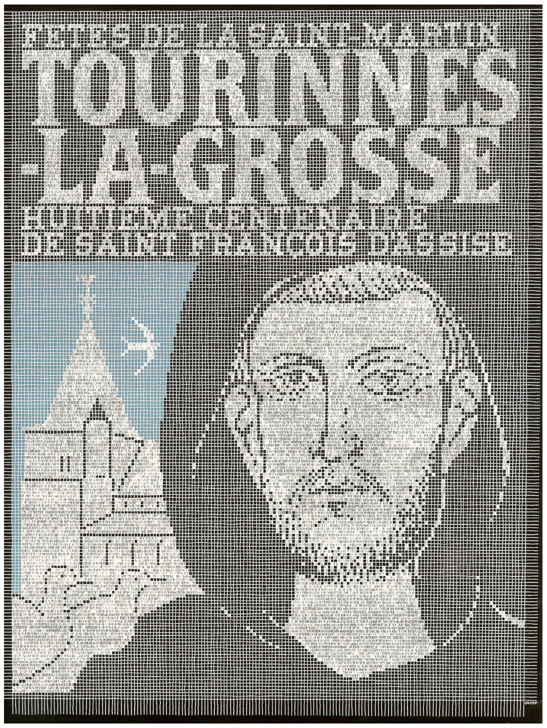


**Figure 29**: Poster by Michel Olyff (1982) inspired by the pixels of embroidery and printers. [Courtesy Michel Olyff]

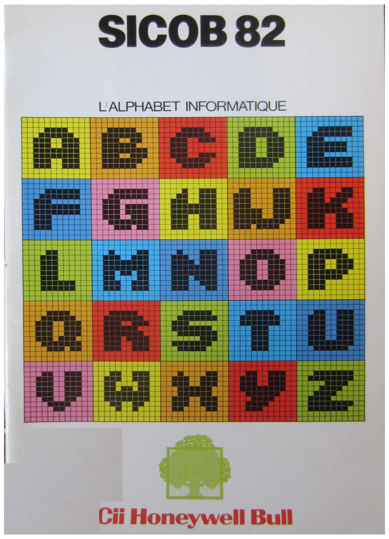


**Figure 30**: Poster of the Honeywell-Bull Company for the SICOB computer show, 1982.

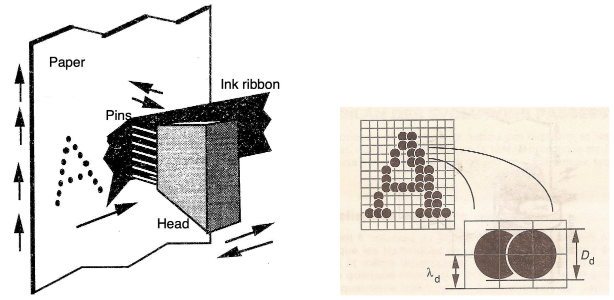


**Figure 31**: Principle of dot matrix printing and construction of characters by overprinting [52].

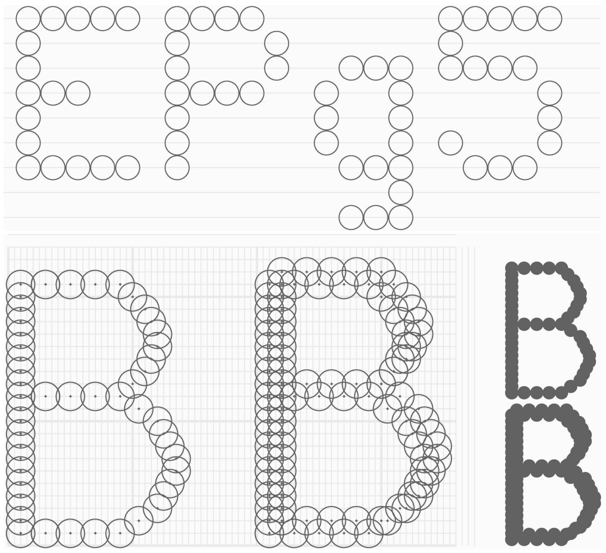
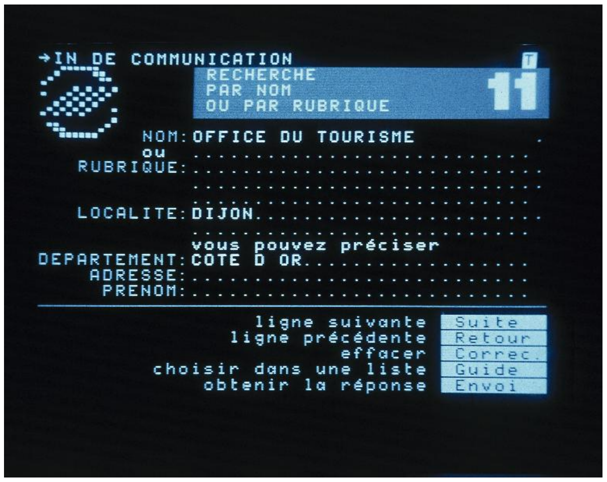


**Figure 32**: Above: classic character printing of a 5 × 7 dot matrix with a 9-pin matrix head. Bottom left: construction of bold on a dot printer using a superposition of two images slightly offset in $x$ and $y$. Right: result of printing a normal B and a bold B. [From a Sanders commercial brochure, circa 1980]

the other of “mail quality”. This higher quality was generally obtained either by the simultaneous passage of the reading head with a slight shift in $x$ and $y$ giving more continuity to the final design, or by using a larger number of pins, or both (figure 32).

Similarly, italics could be simulated by a slant but also by a more adequate design. Unlike daisy wheel printing, it was then possible to use several character styles, weights, etc. in one line without manual operation. Of course, all these dies were drawn by hand (today’s character displays on LED-type lamp panels are made by filling in characters defined by their outlines).

Dot matrix printers were used extensively from the 1970s–1990s as the printers distributed with the early personal computers. The emblematic example remains the 9-pin LaserWriter of the original Macintosh (figure 36). They were dethroned in the mid-1980s by the arrival of laser printers.



→IN DE COMMUNICATION
RECHERCHE
PAR NOM
OU PAR RUBRIQUE

NOM: OFFICE DU TOURISME
ou
RUBRIQUE: ........................

LOCALITE: DIJON........................
vous pouvez préciser
DEPARTEMENT: COTE D OR...................
ADRESSE: ........................
PRENOM: ........................

ligne suivante Suite
ligne précédente Retour
effacer Correc.
choisir dans une liste Guide
obtenir la réponse Envoi

**Figure 33**: A typical Minitel screen, the home page of the French electronic white pages.

## 3.8 Three historical cases

The screens appearing with the first mass-market computers had low resolution which, given the enormous size of the pixels forming the characters, gave typography by computer a bad reputation.

Around 1980, two machines were created which used these bitmapped characters (for screen only or with printer), emblematic of this time: in France, the Minitel and in the United States, the Macintosh. We add here a third example, that of a font which was probably the first one designed for such highly "mosaic" characters, Lucida.

### 3.8.1 The Minitel

Studies on Minitel started in 1979 at CCETT[2] in Rennes (France). The initial goal was to launch a videotex network accessible by a low-cost terminal and then to make an "Electronic Directory" available to all telephone subscribers, which was a commercial and long-lasting success — it was not completely ended until 2012 [97, 93].

The Minitel was a passive computer terminal, consisting only of a keyboard and a screen. The screen (figure 33) was a text matrix with a size of 25 lines by 40 columns. A line of text could thus receive 40 characters, of fixed size as for a typewriter; this is around 1980.

Each character was formed on a grid of 7 pixels in width by 10 pixels in height (a little larger than the $5 \times 7$ of matrix printers). These typefaces were much criticized at the time. For many people (especially typographers), it was the first contact with

[2] *Centre commun d'études de télévision et télécommunications*: Joint Center for Television and Telecommunications Studies

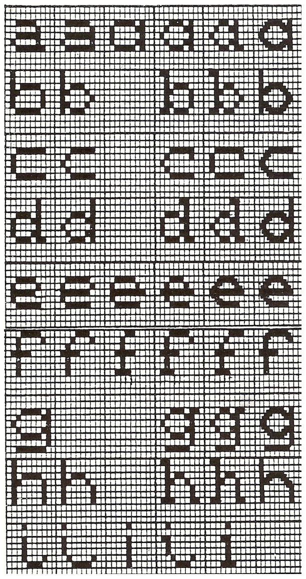


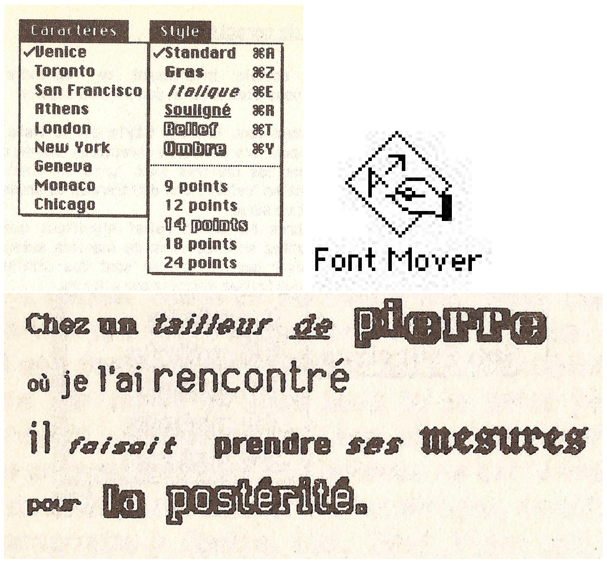
**Figure 34**: The Minitel characters had been tested with several variants, the choice having been made following readability studies [35, p. 56].

computerized typefaces. However, they had been the subject of extensive legibility studies (in the spirit of the work done since Javal, see [26]). Figure 34 shows some of the typeface models used for testing at CCETT.

### 3.8.2 The "original" Macintosh

The Macintosh was the first mass-market personal computer launched by Apple Computer, in January 1984. The project was started at the end of 1978 by Jef Raskin, who wanted to create a computer that was easy to use and inexpensive, and therefore accessible to average consumers. He joined forces with Burrell Smith and then, in 1980, with Steve Jobs who introduced the mouse (which he had seen working at PARC, page 47). In some aspects, notably the graphical user interface, the Macintosh followed the Apple Lisa computer, released a year earlier.

The Macintosh's display device was a 1-bit CRT screen (black and white) with a resolution of $512 \times 342$ pixels. The desktop processing (DTP) standard (which we still find for web images), corresponding to 72 dpi, would come from there. This value is not insignificant. It is close to the 72.27 typographic points per inch of the Americans: 8 points (pixels) of the screen measured thus 8 points (typographic); a character of 12 typographic points was drawn with 12 pixels, including the slope. To this computer, it was also possible to connect a printer with 9 pins, the ImageWriter, designed by the Japanese company Itoh and already in use at Apple. This printer had



Chez un tailleur de pierre
où je l'ai rencontré
il faisait prendre ses mesures
pour la postérité.

**Figure 35**: The first Macs were noted for the quality of their fonts; for the general public, these were the first computerized fonts seen! At left, a menu (composed in Chicago) allowing the user to choose a font, its variant (bold, italic, etc.) and its body; at bottom, demonstration of font combinations [Images from the *Guide Marabout du Macintosh*, 1984; courtesy Susan Kare]. At right, the Font Mover icon, by Susan Kare for the original Macintosh (1982).

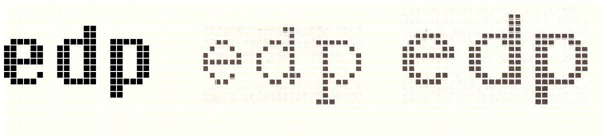
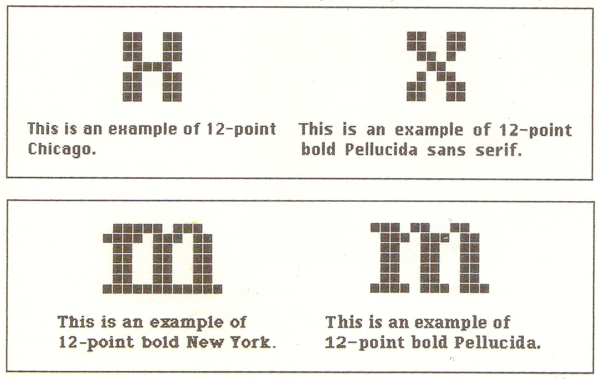


**Figure 36**: Three of the original Macintosh fonts: Chicago (12pt), New York (12pt) and Geneva (14pt). [Courtesy Dafont and Susan Kare]



This is an example of 12-point Chicago.

This is an example of 12-point bold Pellucida sans serif.

This is an example of 12-point bold New York.

This is an example of 12-point bold Pellucida.

**Figure 37**: Bigelow & Holmes showed, with their Pellucida font initially conceived for the VAXstation, that even with the low screen resolution of the original Mac, one could improve the readability of the characters. [*Macworld*, 1985]

| | |
|---|---|
| Chicago | ABC XYZ abc xyz |
| Geneva | ABC XYZ abc xyz |
| Monaco | ABC XYZ abc xyz |
| New York | ABC XYZ abc xyz |

**Figure 38**: The four main fonts of the original Macintosh, redesigned in TrueType by Bigelow & Holmes using Ikarus, from the original bitmaps by Susan Kare [29, 66]. The nominal size shown here is 32 pt. [Courtesy Bigelow & Holmes]

a resolution of 144 dpi. A printed text thus had the same size as its image displayed on the screen, but with twice the resolution (one screen pixel corresponding to four pixels on paper).

This first Mac came with four fonts, all designed by Susan Kare, to whom we also owe nearly all the Mac icons (such as the Font Mover icon, figure 35). These fonts were named Chicago, Geneva, New York and Monaco (figure 36). Chicago was a special case: it was the "system" font used to display the Mac's commands and which existed only in size 12, deliberately a little bold. These fonts were drawn directly on screen by Susan Kare, in a grid, using a small editor designed by Andy Hertzfeld, letter by letter, size by size. Hertzfeld also wrote small programs to distort these characters to make several variants such as bold, italic, shaded, raised, etc. (figure 35), each of which can be combined.

These fonts were usable by the Mac's word processing system, MacWrite, which was one of the first mainstream WYSIWYG applications. Professional typographers, who obviously didn't take this new "typography" seriously, found it hard to accept that a few years later this same Mac would offer typefaces printed with a quality bordering on their tradition.

Let's anticipate the rest of this article a bit... These first bitmapped fonts were redrawn in TrueType by Charles Bigelow and Kris Holmes (figure 38), who used a beta version of the IkarusM software, using only line segments and circular arcs for the curves (like Renaissance drawings, page 35); these were converted into quadratic splines. TrueType Chicago was designed to render bit-for-bit the same (except for a few symbols) as the original Chicago bitmap font, at the system size; the other TrueType designs diverge further from the originals [29].

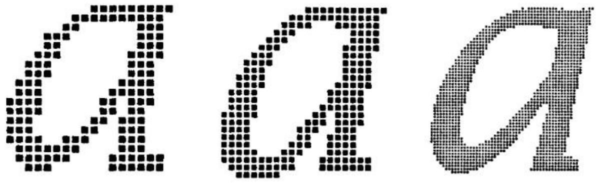


**Figure 39**: Lucida, first released in 1984, was the first typeface design designed for low-resolution printers. The image shows the Lucida seriffed lowercase ‘a’ at three resolutions corresponding to 8, 10, and 24 point fonts on a 300 dpi laser printer. The effects of undersampling (insufficient resolution) are evident. At left, the lowest resolution shows a strongly aliased image with “jaggies” that disrupt the curved and diagonal letter elements. At right, the highest resolution still shows noise along the contours. When these idealized bitmaps are reconstructed as actual images by a laser printer, the sharp images of the stairsteps are smoothed, but some distortion of the forms remain. (Text and images from [30].)

### 3.8.3 Lucida

Although it is a font rather than a computer system, and although it was released later than the previous examples, let us point out that Lucida by Bigelow & Holmes was the first font designed specifically for (not adapted to) low-resolution printers, such as the 300 dpi laser printers of the time (figure 39 and [125]). They also created (by hand) a companion screen font, Pellucida (figure 37).

## 4 Mathematical character models

It was immediately tempting to have the computer do the tedious work of preparing the *run lengths* previously mentioned, and as early as 1965 computer scientists began to write such systems. The basic idea, used almost universally, was to consider that a character is a mathematical surface (defined by its contours) which is projected onto a bitmap and which must be filled. We will first recall that these contours have been known since antiquity, then we will see how they have been improved and adapted to the needs of typography.

In general, these models were based on the existence of an already-drawn character that was to be scanned (Ikarus for example), but some went further by proposing tools to prepare these outlines (e.g. CSD with a modular approach, or METAFONT by using the ductus of calligraphers). Figure 1 showed the chronological evolution of these systems and tools.

### 4.1 Models in antiquity

It is known that during Roman antiquity, since at least the beginning of our era, patterns of lettering were already being used with the ruler and compass. Thus, the capitals of the famous Trajan column (dedicated in 113 CE) have been shown to be rigorously based on straight line segments and arcs of a circle (figure 40).

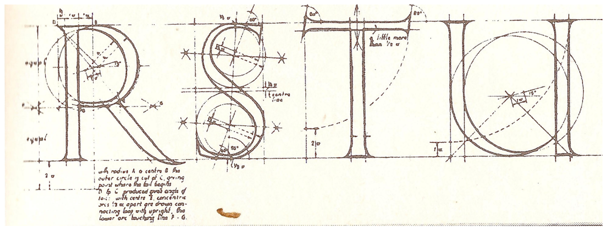


**Figure 40**: The Roman capitals on the Trajan column were drawn with a ruler and compass. Study by Edward Catich, 1968 [42].

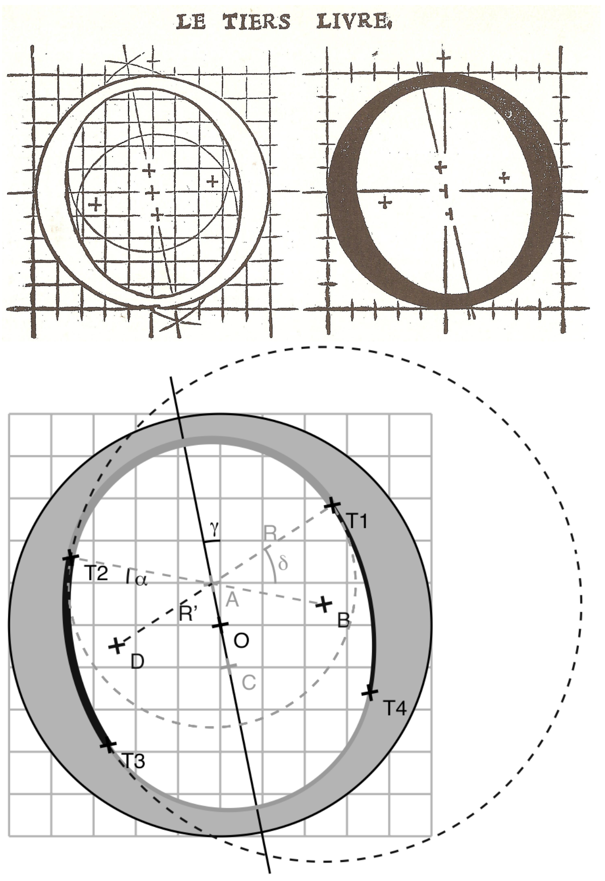
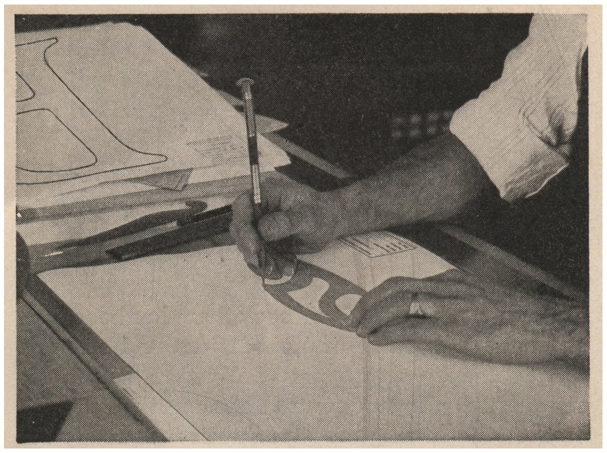


**Figure 41**: Above, the O capital as seen by Tory, *Champfleury*, 1529; below, a trigonometric interpretation [13].

During the Renaissance, various authors proposed models for the letters engraved on the pediments of public or religious buildings based on the Roman capitals and using the only constructions



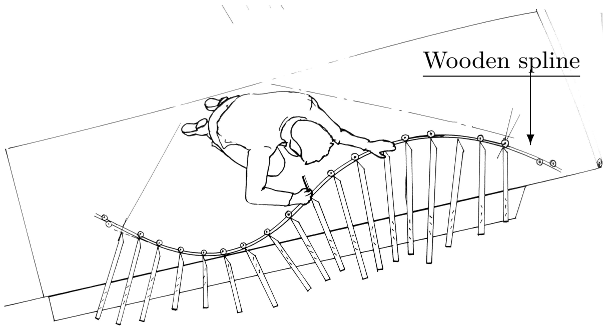


**Figure 42**: Historical tools for complex curve drawing. Top: French curve used when preparing dies for Linotype, around 1930; bottom: wooden spline used in naval carpentry, around 1990 [54].

then known, the ruler and the compass. These models are those of Damiano de Moile, Felice Feliciano, Luca Pacioli, Luca Orfei, etc. They were frequently quoted by typographers [1, 36, 101, 121, 130] or analyzed mathematically [13, 81]. It was Dürer in Germany and then Tory (figure 41) in France, in the sixteenth century, who made the first models substantively applied to typefaces for printed texts, with not only capitals but also lowercase letters. This way of modelling typefaces with a ruler and compass was long-lasting, since it is found in the eighteenth century (e.g. the Romain du roi [15]) and in preparatory drawings by Eric Gill in 1927.

Many professions (carpenters, marine carpenters, architects, industrial designers, road engineers, boilermakers, for automobiles and airplane wings, etc.) have had the problem of drawing harmonious curves passing through a certain number of points but which could not be drawn with the compass in a simple way (that is, without using many arcs of circles). This was solved manually by using tools (figure 42) such as the French curve or the spline (a word that will soon be found again in this article!).

## 4.2 Curves, mathematics and approximation

Mathematical studies on curves were initiated by the Greeks and the Romans and developed at length by their successors, in particular at the end of the 17th and the beginning of the 18th century (Euler, Monge, Cauchy, Legendre, etc.), with the theory developed further in the 20th century (Hermite, Bernstein, ...).

When the equation of a curve is not known, it can be approximated by simpler pieces of curves.

**Lines and circles.** The simplest curves are line segments. This is what plotters did, where the curved body of an R is replaced by five straight line segments (figure 5, right). Less simple curves are circles. This is what Tory did (figure 41), replacing the vaguely elliptical lower curve of his O by four arcs of circles [13].

**Conics.** For a long time, mathematicians looked for curves more complex than straight lines and circles. It turned out that circles belong, together with the parabola, the hyperbola and the ellipse, to the class of conics. It is therefore natural that some font models use conics and in particular parabolas; for example, Coueignoux (page 38 and [43]) and TrueType (figure 70).

**Superellipses and spirals.** Some ellipses have a more rectangular shape, or even the shape of a rectangle with rounded corners: the superellipses (called super eggs by the Danish poet–designer Piet Hein, 1905–1996). Typographers have used them (figure 43). But, what interests us most here is that these curves have also been used to draw pieces of type outlines, for example in the Itsylf (page 38), CSD (page 38) and METAFONT [80] systems.

The kinematic study of road layouts, then of railroads and highways, led to the use of special curves for the connection between two straight segments (change of direction of a road for example). The most "comfortable" trajectory is not a circular arc but a clothoid arc (or Cornu spiral or Euler spiral). These spirals were used in typography by Purdy for the Varityper (figure 44) and some researchers currently recommend the use of such clothoids [90].

**Bézier quadratics and cubics.** Shortly after 1950, when computer graphics started developing, engineers needed to define curves to calculate profiles of, for example, automobile body panels. This simulated what marine carpenters used to do, i.e. to use, as in figure 42, physical splines, which first meant to cut these large curves (now called splines) into smaller pieces. Of course, the small pieces had to joined together while keeping the curve smooth. This is how Pierre Bézier, an engineer at Renault (where

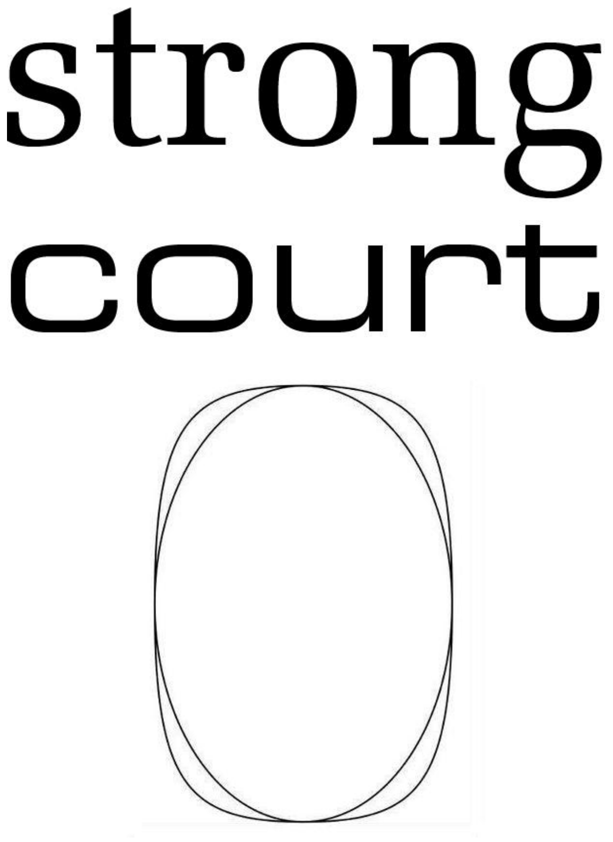


**Figure 43**: Superellipses have been used in type design, notably for ‘O’. Top: Melior by Hermann Zapf (1952); middle: Eurostile by Aldo Novarese (1962) [Courtesy Peter Karow]; bottom: Lucida Grande Mono DK by Bigelow & Holmes (2014) with, inside, an ellipse with the same axes [Courtesy Charles Bigelow].

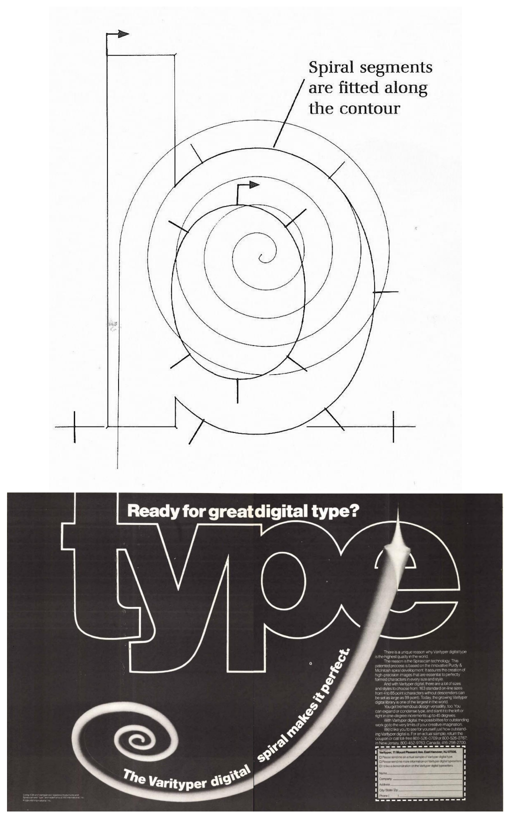
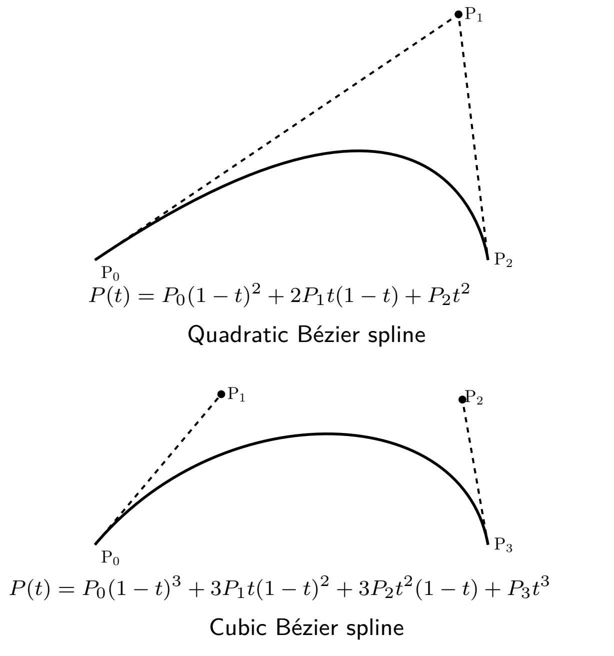

Spiral segments are fitted along the contour
Ready for great digital type?
type
The Varityper digital spiral makes it perfect.

**Figure 44**: Approximation of a character via pieces of spirals. Top: an illustration of the principle (from [70], with permission of Peter Karow);
bottom: an advertisement by Varityper (appeared in *U&lc*, Aug. 1984).



$$P(t) = P_0(1-t)^2 + 2P_1t(1-t) + P_2t^2$$

$$P(t) = P_0(1-t)^3 + 3P_1t(1-t)^2 + 3P_2t^2(1-t) + P_3t^3$$

**Figure 45**: Diagram and formulas of two Bézier curves frequently used in digital typography.

he had already created Unisurf, the archetypal CAD software), studied the curves that now bear his name (Bézier curves or B-splines), which are in fact special cases of Hermite and Bernstein polynomials. To be practically usable, these curves had to be easily and quickly computable. The French mathematician De Casteljau (at Citroën) discovered a very efficient algorithm for plotting based on binary divisions.

Let’s just show here the Bézier curves used in typography and in particular the two most common models, the quadratic and cubic splines (figure 45). Their names derive from “quad” (square, therefore two) and cube (three), terms with which, since the Renaissance, mathematicians named the powers of two and three.

The cubic Bézier curves (figure 45, bottom) are defined by four points P0, P1, P2 and P3, i.e. by the two tangents P0–P1 and P3–P2. The curve passes through the points P0 and P3 and is included in the parallelogram P0–P1–P2–P3, which gives a first approximation. Introduced in the typographic world by Xerox, and made widely known by Adobe, they are used by many font formats.

Quadratics (figure 45, top) are defined by the triangle P0–P1–P2 and are in fact pieces of parabolas. They are also used in some font formats, notably TrueType (see figure 70). They ensure the continuity of tangents at junctions, but not those of curves, and therefore need to be divided into smaller segments than cubic curves. Thus, although each quadratic piece inherently takes fewer points to define than a corresponding cubic, more pieces are needed. When converting from quadratics to cubics, the curves are not quite equivalent.

## 5 First contour-based fonts

The trial and error period of the 1960s is not well known. But, thanks to Lynn Ruggles [112], we can mention a few names.

**ITSYLF, the first font generation system, 1968.** Ruggles considers this to be one of the very first systems specifically dedicated to type design, although it was never made operational. This system, ITSYLF [100], developed by Mergler and Vargo, had two novel features compared to the few existing systems. First, it used conics, more precisely superellipses (or Lamé curves), rather than arcs of circles, to approximate the curves. The arcs of curves are defined by superellipses of the form $(X/A)^F + (Y/B)^F = 1$. For two values of $A$ and $B$, we can vary $F$ and obtain a series of more or lesser curved lines passing through these two points $A$ and $B$.

Second, it does not define a font, but a family, thanks to a skeleton and parameters allowing to refine the final shape. These parameters are defined for the E, and then adapted automatically to the other letters. Figure 46 shows the skeleton of a C and variations calculated automatically for this C by varying the parameters. The whole font would thus always be homogeneous.

This is not far from what will be possible a few years later with METAFONT (page 41).

**CSD, FRANCE and Coueignoux' works, 1973.** The Frenchman Philippe Coueignoux developed, in 1973 at MIT (USA), what Knuth considers "the first use of sophisticated mathematics to describe letterforms by computer" [82]. His work was mainly published in his two theses, *Compression of Type Faces by Contour Coding* in 1973 and *Generation of Roman printed fonts* [43] in 1975, the latter being widely cited: even though his "academic" research did not lead directly to industrial developments, his ideas are found in many later systems.

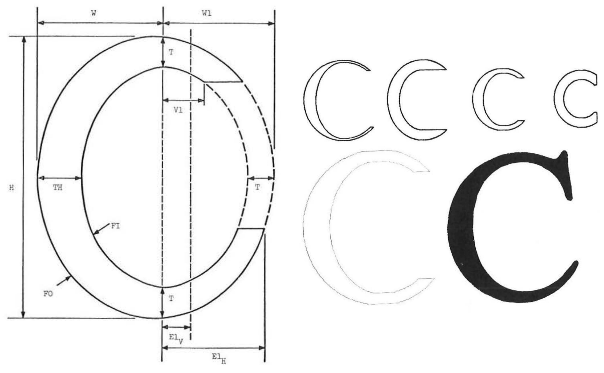


**Figure 46**: The ITSYLF system of Mergler and Vargo (1968). On the left, the schematic of the C, with indications of the parameters (W, W1, T, V1, ...). On the right, top: letters printed by varying these parameters; bottom: from the basic C, the C of Times Roman can be defined. [Courtesy *Visible Language*]

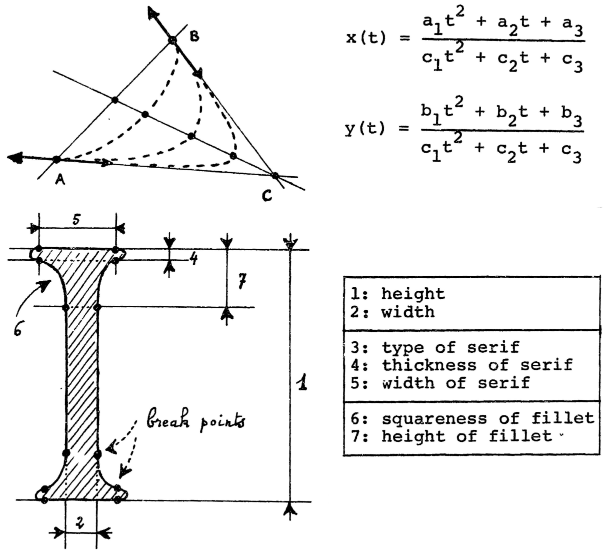


**Figure 47**: Coueignoux's use (in 1973) of conics: the curves (in dotted lines) are defined by the four coordinates of the points A and B and by the distance from C to the curve. Bottom, examples of parameters to define a character. [43]

Coueignoux [43] describes a model for encoding character outlines, which he developed apparently without knowing the concurrent work of Karow (which we'll discuss subsequently). As in ITSYLF (though without knowing it either), Coueignoux uses line segments for stems, bars, etc., and conics for the arcs of terminals and superellipses for bowls. Similarly, characters are defined using parameters.

What is new is that Coueignoux uses a structured grammar (like those of Chomsky, well known to academic linguists and computer scientists since

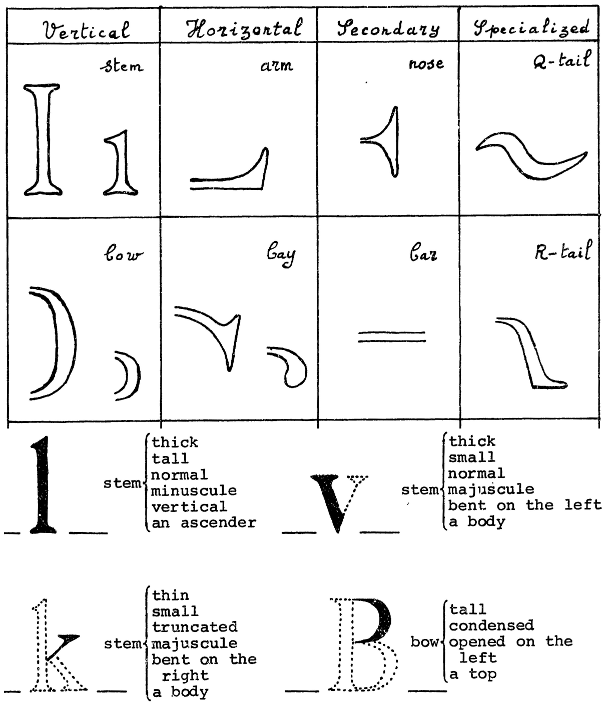

Vertical
Horizontal
Secondary
Specialized
stem
arm
nose
Q-tail
bow
bay
bar
R-tail
stem {thick tall normal minuscule vertical an ascender
stem {thick small normal majuscule bent on the left a body
stem {thin small truncated majuscule bent on the right a body
bow {tall condensed opened on the left a top

**Figure 48**: Some basic CSD primitives and extracts from the generic grammar. [43]

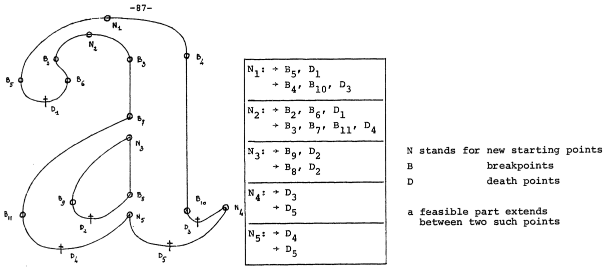

-87-
N₁: → B₅, D₁ → B₄, B₁₀, D₃
N₂: → B₂, B₆, D₁ → B₃, B₇, B₁₁, D₄
N₃: → B₉, D₂ → B₈, D₂
N₄: → D₃ → D₅
N₅: → D₄ → D₅
N stands for new starting points
B breakpoints
D death points
a feasible part extends between two such points

**Figure 49**: The FRANCE software, by Coueignoux in 1975, makes it possible to find the breakpoints of splines and segments. [43]

about 1965) to define his characters. *Primitives* form the basic elements allowing to define characters. This is an incremental definition of characters, of which we find new attempts since the beginning of our third millennium [67]. This generic method is associated with a font production system, CSD (*Character Simulated Design*), as shown in figure 48, bottom.

Moreover, the FRANCE system (*Font Retrieval: A Natural Coding of Edges*) is a program that does what is now called *autotracing* (like Ikarus, as we will see): starting from a character known by its representation in the form of a matrix of points, it deduces an algebraic description, using splines (figure 49).

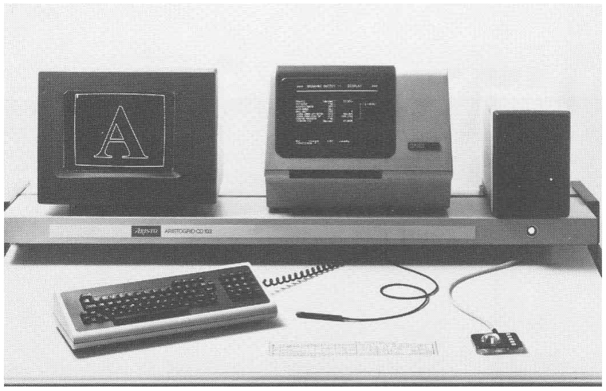


**Figure 50**: The first Ikarus hardware, 1973. [73]

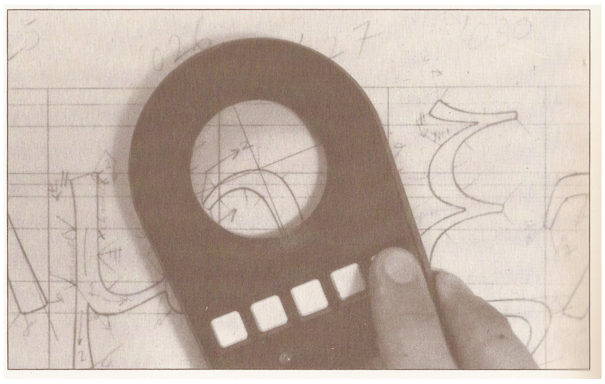


**Figure 51**: Using the Ikarus "mouse"; here in the context of the Euler project at Stanford (1983), led by Bigelow, Knuth, and Southall, with characters designed by Zapf. [119]

## 6 Ikarus

URW (named after its first two founders, Rubow and Weber) was established in 1971 in Hamburg, Germany. Peter Karow joined the company in 1972 and was responsible for the automation of the production of fonts for photocomposers. See his background and his research and development in [68, 71, 72, 73].

Peter Karow's first "customer" was Walter Brendel [73], a type designer who was responsible for fonts such as Lingwood and Volkswagen and who was then starting a digital type library for photocomposers (this was only around 1970; it became the basis of the "TypeShop Collection" of Elsner+Flake). His customers wanted modifications to his fonts, such as "blacker", "spaced out more", "shaded", etc., which could not be done on the bitmaps and had to be done from the drawings themselves. Karow first made tests with characters cut from 15 cm high vinyl plates (quite similar to Frutiger's scratch cards) and then understood the interest in digitizing the characters to work with reusable formats.

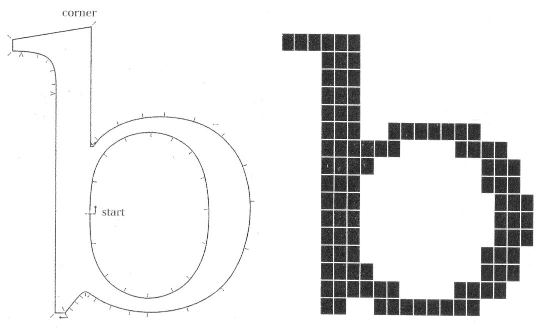


**Figure 52**: Left: outline of a 'b' with its guide points for Ikarus; right: the same after translation from IK format to bitmaps, here at low resolution. [70]

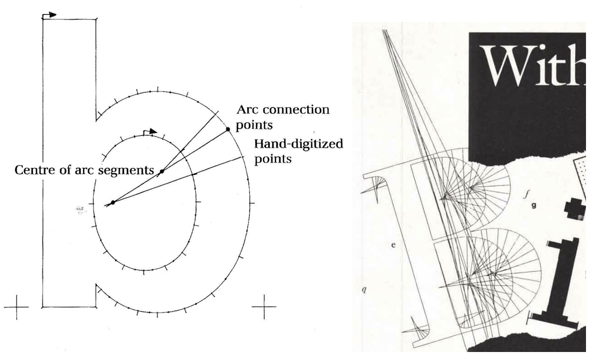


**Figure 53**: Left: approximation of a 'b' by arcs of circles [56]; right: Bitstream advertisement based on the construction of a 'b' by arcs of circles [*U&lC*, vol. 12, no. 4, Feb. 1986].

Karow substantially started his project at the end of 1972, in collaboration with Aristo, a CAD company from Hamburg, who was responsible in 1960 for the Perthronic table which drew stick letters (page 24), and later for the Aristogrid tablet with cursor. He named it Ikarus in May 1973. The commercialized system included (figure 50) a digitizing tablet controlled by a cursor, cousin of the mouse (figure 51), and a computer with CRT and alphanumeric screen, keyboard, etc. Ikarus was written in Fortran and was quickly adapted to the VAX and Sun minicomputers on which many professionals used it. (These workstations were called "mini" in comparison to the large computers of the time, but were still almost entirely used only by companies and had nothing to do with personal computers.)

Ikarus was, first, a system for digitizing character drawings. By clicking — with the cursor box moved over the enlarged character on the tablet — on a starting point, angles (or corners), points distributed *on the curve* (and in particular the ends of the curves and the points of inflection) and tangents (with the help of two points allowing measurement of the angle) for breaks in continuity, one obtained a set of coordinates, stored in a format named IK [68]. The system analyzes these points and deduces a mathematical description of the contour of this character using cubic splines. The first applications of this system were to calculate directly the bitmaps, or more precisely the run lengths, making it possible to control a photocomposer and in particular the Digiset (page 28).

The next applications of this system were for plotters that only worked with line segments and arcs (page 23). When using this format to draw the outline of such a character, the IK format was then transformed into another format, DI, describing the outline with vectors and circular arcs. The center and radius coordinates were calculated by Ikarus based on the points of the IK format [68].

Early Ikarus customers also used this DI format, either locally (e.g. for arches, Bigelow, page 34) or systematically for all curves (this was the case for the early Bitstream fonts, figure 53 and [6, chap. 6]).

In the 1980s, URW played an important role as a font vendor and, following the advent of PostScript, improved Ikarus.

## 7 Filling, rendering and hinting

### 7.1 Bitmaps and filling

Filling of characters which are defined by their contours requires calculating the zones of the bitmap which will be scanned, line by line, by the laser beam (which comes back to calculating the run lengths of the photocomposition, see page 28), taking into account the theoretical contour. We reuse the scan conversion techniques developed since the 1960s for computer graphics. The principle is to follow the curve line by line and mark the pixels whose centers are inside the contour delimited by the curve (figure 54). This method was adapted to characters in the 1980s by researchers such as Ackland, Bétrisey, Gonczarowski, Hersch and Pavlidis [60, 109]. Similar methods are used as well with METAFONT84 and described in [57, appendix F.1.4] and [83, chapter 24].

The difficulty is to determine the angles (e.g. vertex of an A) and not to fill in extra or omit some pixels (dropout) because of singularities of the curve. Bad detection of such points explains why some printers of the 1980s erroneously drew horizontal lines across the whole page, the point of a V, for example, having been badly detected.

These filling operations are not independent of those of rendering and hinting (discussed below)

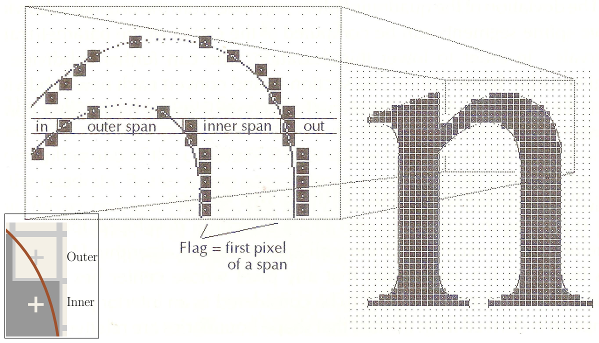


**Figure 54**: Principle of the filling of a character by marking the limits of the internal zones (inner span) and external (outer span) of the zones to blacken in a bitmap on which one projects the theoretical curve of the character [60]. [Courtesy Roger Hersch]

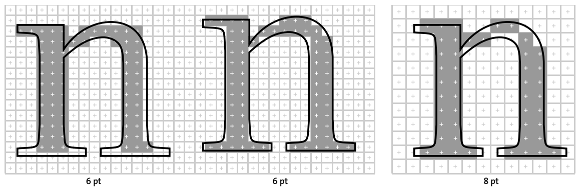


**Figure 55**: For the same glyph (with thick black lines), the blackened pixels depend on the definition (here, pixel size of 6 pt and 8 pt) and on the precise position of this glyph in the grid.

which allow more precisely refining the choice of the pixels to blacken.

### 7.2 Rendering improvements, hinting

The filling algorithm does its job well, but the results depend on the resolution (see figure 55 for cases of 6 pt or 8 pt pixels) and also on the position in the grid (again figure 55, the two 6 pt cases, and figure 56). Many rendering defects appear in this way, for example unevenness of descenders, disappearance of thin parts (serifs, ties, swashes), appearance of holes, etc.

This phenomenon is only noticeable for small and medium sizes and low resolutions (< 200 dpi, which is (commonly) the case for screens and for the first laser printers). To limit these phenomena, many fonts have been designed specifically for specific screen sizes (e.g. sizes 10 to 12). Let us cite Verdana by Matthew Carter (1996), the typefaces Base 9 and Base 12 from Zuzana Licko (1995) and Hachette Multimédia by Olivier Nineuil (1996).

The Ikarus system, and those that we will see later on (METAFONT, Fred, etc.), had all the information to prepare the bitmaps; the basic idea being to make (very slight) shifts in the theoretical curve

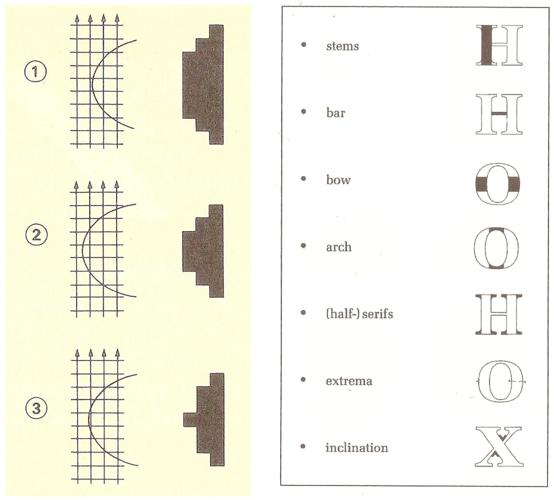


**Figure 56**: Examples of needed rendering improvements (in 1 and 3 the top of the curve is too close to a pixel border: the result is bad) and, right, some of the cases studied by Karow since 1981. [68]

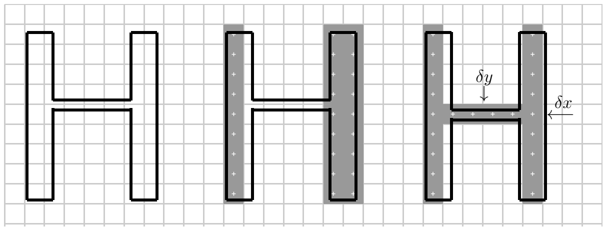


**Figure 57**: Simple example of hinting. Left: the outline of an H; center: how it would normally be translated into pixels; right: displacements, downwards for the bar and backwards for the right stem, give a better rendering.

so that the filling would be more in the spirit of what is expected (figure 57). But with the appearance of PostScript, or rather raster image processors (RIP), we will see that it is more complicated.

There is no perfect solution to this problem even today, but many approaches have been made. Descriptions of these methods can be found for example in [57, 60, 70, 109] and, recently, some manufacturers' websites explain their methods. The difficulty lies in the way to express what one wants to do in a language (or by a method) accessible to the type designer, who is the only one competent for these drawing problems. Hinting methods are thus characteristic of font systems, whether Ikarus, Type 1, TrueType or OpenType.

## 8 TEX and METAFONT

Around 1975, the American mathematician Donald Knuth was revising a volume in his magnum

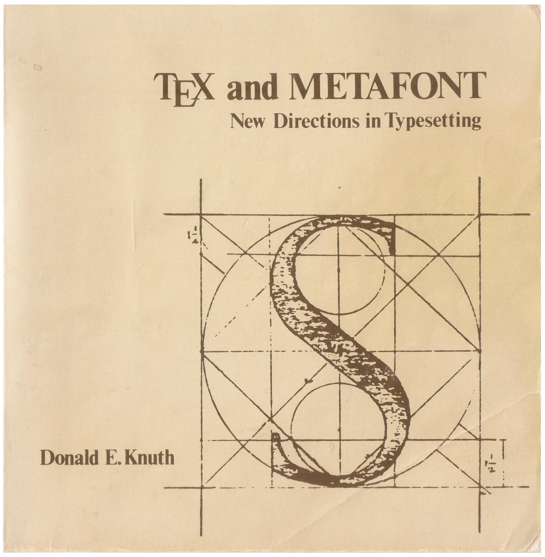


**Figure 58**: The book in which Donald Knuth first presented (1979) both TEX and METAFONT. [80]

opus, *The Art of Computer Programming* (in brief, *TAOCP*), and realized that the typesetting and printing methods that had become prevalent — not only the composition of mathematics but also that of text — were falling far short of the quality of his earlier editions. He then embarked on the adventure of building himself a typesetting system using digital fonts for which he also defined a construction tool. This is the TEX+METAFONT "couple", with Knuth's first publications in 1978–79 [78, 80] (figure 58), which *TUGboat* readers know well!

In the 1970s, there were few text editors (see e.g. [55, 108, 124]) and they were devoted to dedicated devices with hardwired fonts: typewriter terminals, line printers, and, rarely, second generation phototypesetters (only n/troff, the Unix documentation language from Bell Labs, can be cited). Furthermore, researchers were in the early stages of mathematically defining digital fonts (new tools such as CSD and Ikarus were still confidential). So Knuth felt obliged to create something new! For a general view of this story, see [22], and let's mention right away that Donald Knuth has published extensively about TEX and METAFONT (see in particular [80], [82] and [86]). Many practical aspects can also be learned in *Fonts & Encodings* by Yannis Haralambous [57].

For his TEX system, originally intended for works including significant mathematics, Knuth also needed a font for mathematics. He decided to design such a font himself and soon realized that he needed a program to design not just characters, but entire fonts and even font families. This is how METAFONT was born, around 1979.

Since the primary purpose of TEX and METAFONT was to typeset *TAOCP*, it was necessary to test METAFONT output with something better than the impact printers then in use. Fortunately, Stanford University had a copy of the first raster printer, Xerox's XGP (see section 9.3). As early as 1977, Knuth was able to use it, thanks to drivers written by Frank Liang and Michael Plass (two of his Ph.D. students). Around 1980, David Fuchs (another of his students) implemented the new concept of device-independent drivers (DVI), including one for the new Versatec chemical printer used then in the TEX project. This was before the first laser printer (see page 48).

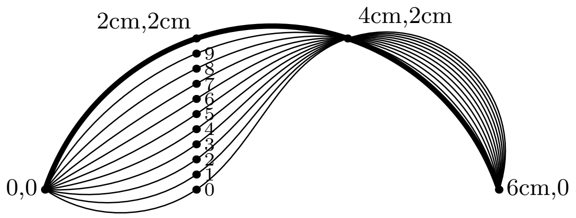
## 8.1 Basic principles of METAFONT

Knuth embarked on an historical study of typography. He learned that, unlike the hot metal types made with a Benton pantograph (and also unlike phototypesetter fonts with optical lenses), traditional movable types did not use geometric scaling. For example, the glyph of an A at body size 16 pt is not just two times larger than the same A at body size 8 pt. Reasons can be due to human vision, printing details (such as ink spreading), etc. That means that a model for font has to use mathematical variables to be general. His study of old type models [81, 86] (as well as the references cited earlier, page 35) showed that characters may be mathematically defined by curves. Furthermore, from studying calligraphy, he discovered the importance of the *ductus* (the shape and order of the strokes used to compose letters) to draw characters, and that a letter can be defined as the trace of a pen moving (along the ductus).

Unlike Ikarus and other products that proposed tools to digitize existing types (at least as sketches), Knuth wanted METAFONT to enable creating a character family from scratch, and without the user having to know any underlying mathematics. Typically, a user says "I would like a nice curve crossing points `(0,0), ..., (6,0)`" and METAFONT chooses the best spline (figure 59). This is why METAFONT is categorized as a declarative programming language (about which we will say nothing further, as beyond our scope here).

## 8.2 METAFONT79, and experimentation

Here we appended "79" to METAFONT since Knuth published the first version of METAFONT in 1979. (He first produced machine-drawn letters in 1977, writing directly in the SAIL language.) He developed METAFONT79 concurrently with the first version



```
% thick line:
draw (0,0)..(2cm,2cm)..(4cm,2cm).. (6cm,0);
% bunch:
for a=9  step -1 until 0:
 draw (0,0)..(2cm,a*.2cm)..(4cm,2cm).. (6cm,0);
endfor
```

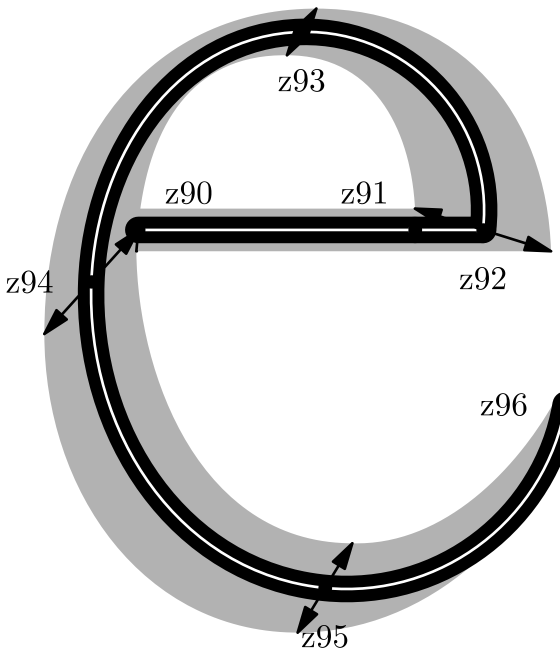
**Figure 59**: This bunch of curves (top) was produced by the given METAFONT79[4] program (below). (Coordinates are added for readability.)

of his Computer Modern typeface (January 1980) with the help of renowned calligraphers and typographers, such as Charles Bigelow, Matthew Carter, Kris Holmes, Richard Southall, and Hermann Zapf. Their remarks, as well as those of his computer science students (especially John Hobby, who designed the key algorithms of METAFONT [63]), led Knuth to completely revise the METAFONT language and program, which he first released in 1984.

During those years, 1980–1984, all of the above, along with many others such as Vaughan Pratt, Lynn Ruggles, John Seybold, Gerard Unger, et al., were involved with the Stanford Digital Typography Program of that time, a set of lectures, seminars, workshops, etc., dedicated to a mix of mathematicians or designers. They strongly influenced METAFONT (see [22, p. 89]). When it's necessary to distinguish between METAFONT's two major incarnations, we specify METAFONT79 or METAFONT84 below.

The main goal of METAFONT79 was to produce all the bitmaps of each font of a family, at all expected body sizes. That implies the strong use of parameters and variables. In figure 59 you can see that the abscissa of the second point is defined by using a parameter, `a`, as a scaling factor `(2cm, a*.2cm)`, i.e. this abscissa is not a numeric constant but a variable. In practice, these variables may represent the body size, boldness, or any other dependencies (for practical examples of such parameters, see figure 63 and figure 64). This is immediately clear for programmers; however, if you look at any glyph description in graphical tools such as Fontographer, Fontforge, Fontlab, and Glyphs, you'll see that coordinates of glyphs are always constant.

[4] In this figure, and in forthcoming ones, METAPOST has been used instead of METAFONT79 or METAFONT84: these languages are, in that case, almost equivalent.



```
epen ...; %grey:
draw z90--z91--z92..z93..z94..z95..z96;
cpen 20; %black
draw z90--z91--z92..z93..z94..z95..z96;
cpen 2;   %white
draw z90--z91--z92..z93..z94..z95..z96;
```

**Figure 60**: Three stacked e characters designed with METAFONT, using the same ductus (the path `z90 .. z96`). The black one and the white one are each painted with a circular pen (a round brush) with diameters of, respectively, 20 and 2; the grey one uses a more complex pen (marked here with black double arrows): its length and orientation depend on the position along the path.

One of the characteristics of METAFONT is that variables may be defined with geometrical equations. For example, the intent in a design that the three stems of an 'm' are equally spaced horizontally might be expressed as

$x_2 - x_1 = x_3 - x_2$

if points 1, 2, and 3 are at the bottom ends of the three stems; whereas the intent that they all end on the same vertical position would be

$y_1 = y_2 = y_3$.

The principal objects handled by METAFONT are the splines that the user may define with a `draw` instruction, and the points of the plane where the spline goes; see the example in figure 60.

Unlike almost all other digital font systems, METAFONT79 does not offer the user a way to describe the characters by their outlines but used only a pen metaphor for drawing glyphs: it assumes their definition via the ductus of a polygonal or elliptical pen, as done by calligraphers and the early printers. Figure 60 shows a nib (white line) which starts from point z90, goes straight to point z92 and arrives at point z96 after having drawn a Garamond-like e.

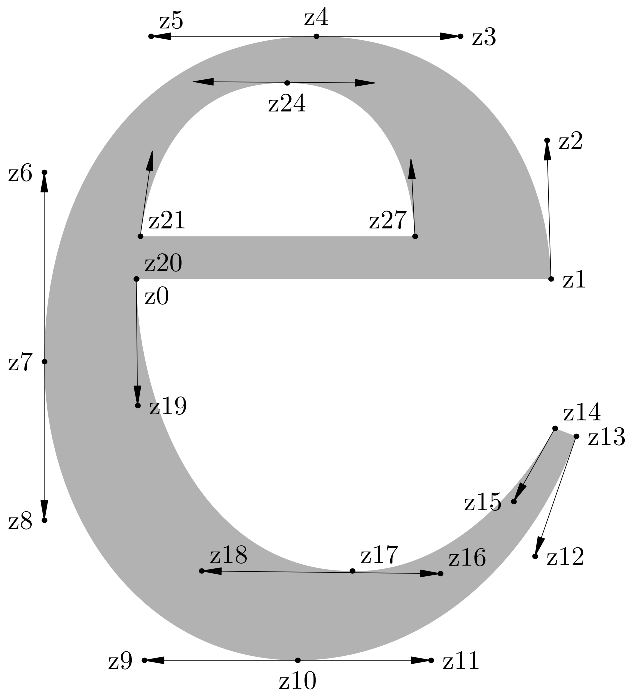
METAFONT79 allows defining curves more precisely, e.g. by defining angles at some points. Various kinds of pens are supported in METAFONT79: circular with various diameters (e.g. the white and black curves in figure 60), elliptical pens, and more general pens that have to be mathematically defined. Erasers are special pens that erase some part of a previous painted area.

The tools we have just mentioned (curves, pens) are those from the user's point of view. Internally, it was a different situation altogether. Knuth and his students used sophisticated mathematics to determine the curves finally drawn or painted. Let's summarize by saying they used polynomial curves, including cubics (Knuth does not use the word "Bézier" in [80] — probably because it was not fashionable at that time!).

### 8.3 METAFONT84

As we mentioned earlier, the people testing METAFONT79 found it was difficult to be used as a design tool by non-programmers, and Knuth completely redefined METAFONT [82], notably with the help of John Hobby [63]. Among the new added concepts, Bézier curves are now intensively used, both internally and from the user's point of view. Characters may be defined by their outlines (described with control points) and related instructions to fill the surface they define. This can be explicitly used by type designers as in PostScript (see below, page 49): figure 61 shows the same e of figure 60, but defined with Bézier control points (you may compare the syntax with that used by Ti*k*Z, `ctan.org/pkg/tikz`.)

However, METAFONT always focuses on the use of pens to draw characters, the creation of Computer Modern being the primary goal. Thanks to new procedures (`pickup, penstroke, penpos`, etc.), it is possible to define new types of pens, their local positioning, their paths, etc. The variation of the pens' marks, together with the use of parameters, makes it possible to draw a whole family of characters at once. For example, figure 62 shows that the arches of an 'n' are defined (without any reference to outlines) by variable pen positions, here `penpos` *i*.



```
z0=...; ...  z27=...;
fill z0--z1 ..controls z2 and z3 ..z4
     ..controls z5 and z6 ..z7
     ..controls z8 and z9 ..z10
     ..controls z11 and z12 ..z13--z14
     ..controls z15 and z16 ..z17
     ..controls z18 and z19 ..z20
     --cycle withcolor .7white;
unfill z21 ..controls z22 and z23 ..z24
     ..controls z25 and z26 ..z27--cycle;
```

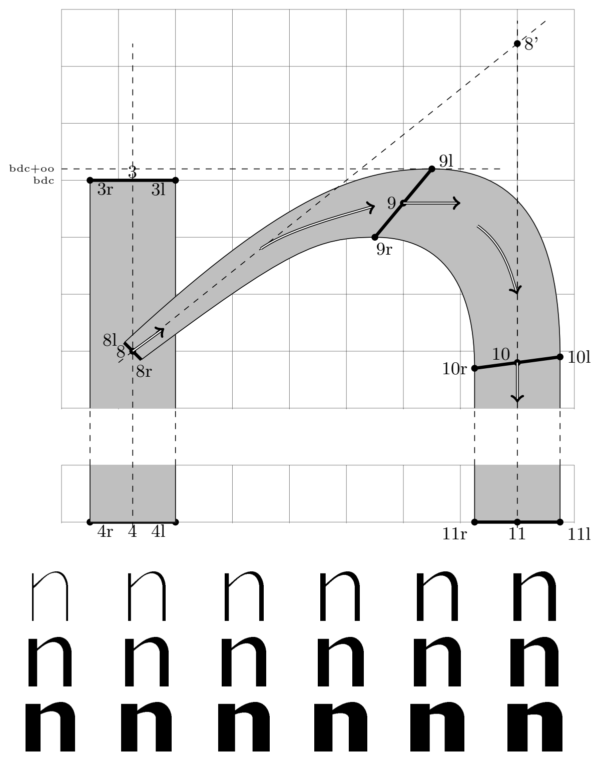
**Figure 61**: METAFONT84 allows painting a character from outlines described as Bézier curves. Compare with figure 60. (Labelled dots and tangents are added for convenience.)

More complex examples are in the final Computer Modern fonts (see section 8.4).

As a programming language, METAFONT offers many possibilities; let's just quote here the fact that "definitions" (also called "macros") allow, for example, making serifs compatible to each character of a font, like the incremental primitives of CSD (figure 48) or like, today, making a serif font with Glyphs using "corner components".

### 8.4 Computer Modern and others

The first large typeface family defined using METAFONT was Computer Modern (also called "cm"); the design was based on Monotype Modern 8A. It was created by Donald Knuth himself, with advice and assistance from Hermann Zapf, Charles Bigelow and Richard Southall. It was in fact the first "total



```
weight=(i/63);          % weight = function of i
loose=...
z3=...; z4=...;                  % left stem ends
penpos4(weight,180);             % pen # 4
penstroke z3e..{down}z4e;        % stroke left stroke
z11=...;
penpos11(weight,180);                   % pen 11
z8=...; penpos8(loose,angle(...)); % pen 8
y9l=bdc+oo; x9=.68[x8,x10];             % point 9
penpos9(.87[loose,weight],-136);        % pen 9
%
penstroke z8e..z9e..z10e---z11e; % arch and stem
```

**Figure 62**: METAFONT allows the construction of a typeface family based on the ductus alone. Above, the principle; below, a selection of 18 n's with different thicknesses and, below, an extract from the METAFONT program. After Haralambous [57], with kind permission.

```
cmchar "The letter e";
beginchar("e",7.25u#+max(.75u#,.5curve#),x_height#,0);
italcorr .5[bar_height#,x_height#]*slant+.5min(curve#-
1.5u#,0);
adjust_fit(if monospace: .25u#,.5u# else: 0,0 fi);
numeric left_curve,right_curve;
left_curve=right_curve+6stem_corr=curve if not serifs: -
3stem_corr fi;
...
path testpath; testpath=super_arc.r(2,3) & super_arc.r(3,4);
y1'r=y0r=y0l+.6[thin_join,vair]; y1'l=y0l; x1'l=x1'r=x1;
forsuffixes $=l,r:
 x0$=xpart(((0,y0$)--(x1,y0$)) intersectionpoint testpath);
endfor
fill stroke z0e--z1'e; % crossbar
penlabels(0,1,2,3,4,5); endchar;
```

**Figure 63**: Definition of the Computer Modern character e in METAFONT84 [84]. The variables such as `bar_height, x_height, monospace` are defined in a driver file, given values as desired for a particular font.

typography pack" [37], since it includes not only roman, italic and bold combined, but also variants of (real) small capitals, serif and sans serif typefaces, fixed width typefaces, and more. Not to mention a very large number of mathematical symbols [84].

All these typefaces have a family resemblance, and for good reason: they are defined by a single METAFONT program with many parameters. Knuth defined about sixty parameters (figure 64) to generate all these fonts; the entire family is completely described in a whole book, *Computer Modern Typefaces* [84]. Figure 63 shows a part of the definition of the model for the e's.

Although METAFONT is reputed to have been little used, hundreds of fonts have been created with it (an "incredible list" has been compiled by Luc Devroye [49]), especially for languages with non-Latin alphabets (Unicode did not exist for quite a few years after TEX and METAFONT), and in particular for ancient languages (including full accented Greek).

Richard Southall used METAFONT again to create Colorado (figure 65) by Ladislas Mandel [116]. This font, intended for the composition of telephone directories, required character to remain very readable even at very small body sizes.

## 8.5 METAFONT and type design

As we mentioned (section 8.2), many type designers have evaluated METAFONT, both during its development, and after. We previously discussed Hofstadter's answer [65] to Knuth's "Concept of a metafont" [82]; see also, for example, [32, 31]. Here are some highlights.

**Family of fonts, parametrization** One of METAFONT's strong ideas is to draw a whole family of fonts and not just one font. To express these instructions, parameters, etc., the designer must express them in the METAFONT language, which is in fact a programming language. And this is the problem

Some type designers, such as Gerard Unger [122, 123], did not fail to say "Besides being a designer, I have no objection to acting as a system operator; but I don't want to become a programmer — let alone a parameterizer."

**WYSIWYG or not** Not all designers are ready to program, as they are used to working on character images and not on how to obtain these images. At a low level, let us say that it is easier for them to drag a dot on a screen and see what happens to some outline than to change the coordinates of this point in some program, experiment with the effect, running the program each

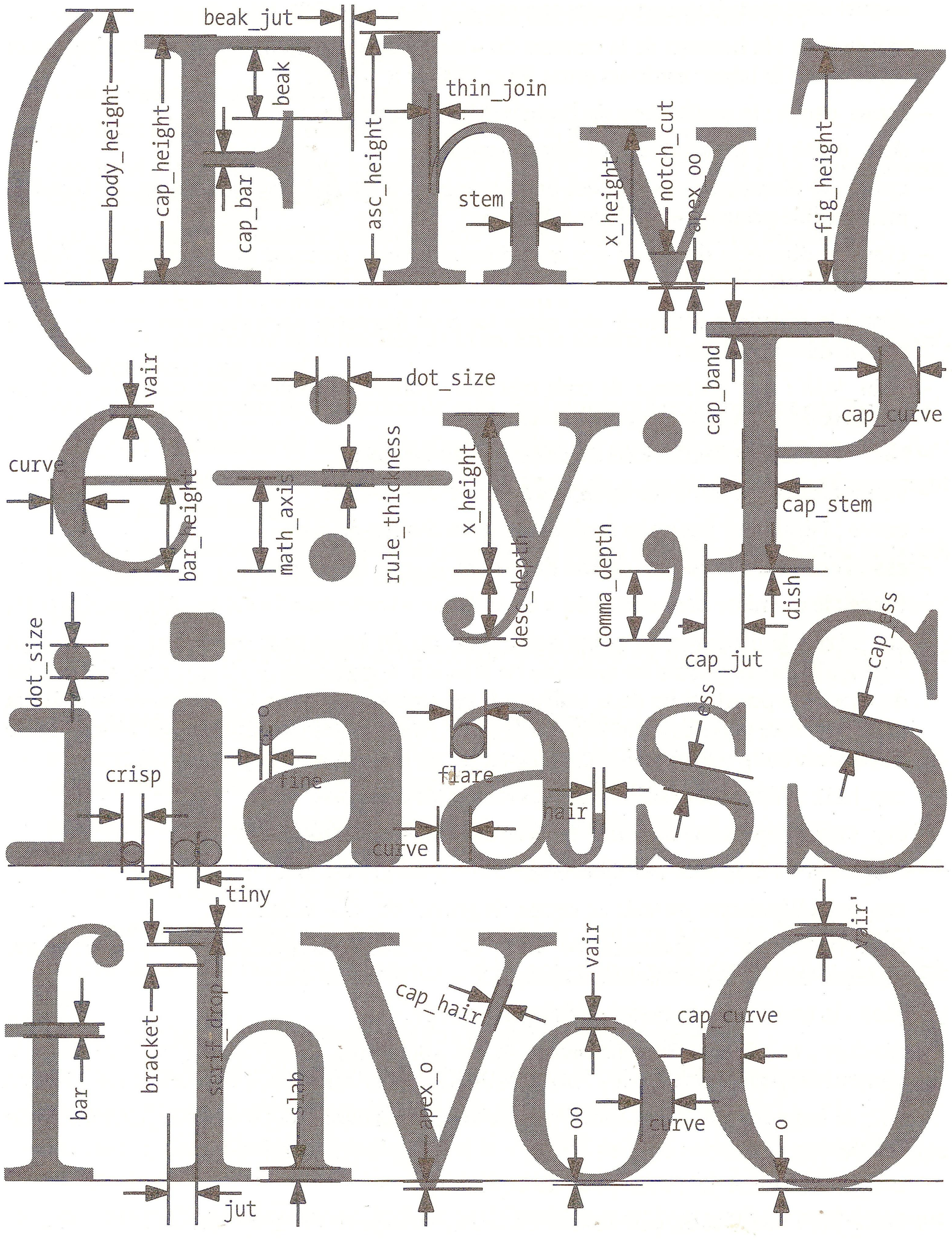
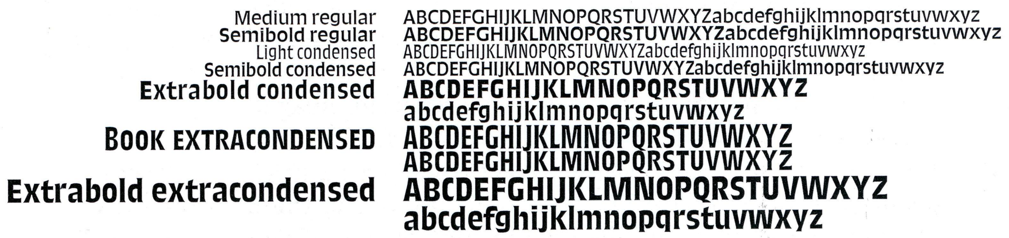


**Figure 64**: The 62 parameters that define Computer Modern, shown via selected characters [57].



Medium regular ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
Semibold regular ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
Light condensed ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
Semibold condensed ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
Extrabold condensed ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
BOOK EXTRACONDENSED ABCDEFGHIJKLMNOPQRSTUVWXYZ ABCDEFGHIJKLMNOPQRSTUVWXYZ
Extrabold extracondensed ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz

**Figure 65**: Specimen of Colorado, a font for telephone directories by Mandel, designed with METAFONT by Richard Southall [117]. Real size; the first four lines are body size 6 pt, with 0.5 point leading. [Courtesy Kris Holmes]

time, until achieving the desired outline. Richard Southall [115, 117] and Dave Crossland [44] have studied this issue extensively.

**Curves** METAFONT84 uses Bézier curves, namely cubic splines. It seems Knuth considered these splines to be, mathematically, nicer than others (including conics). However some designers (at least ones used to Ikarus and later TrueType fonts) prefer to use quadratic splines as they provide (require) more points to control, and these points are closer to the expected outline, so more controllable. See figure 70.

**Ductus model** The ductus model (and the related concept of pen in METAFONT) is surely good for calligraphy or Oriental scripts based on separate strokes. However, this concept was largely abandoned for type design since the time of Aldus Manutius (around 1500), this abandonment being the precise differentiation between calligraphic writings and typographical ones. Since that time, typographers see types as surfaces, for which outlines are everywhere. Alas, the surfaces are less suitable for parameterizing, for example, the boldness of the arches.

## 9 Xerox PARC

The American company Xerox was founded in 1940 to exploit a new photocopying process, xerography, and quickly became the world leader in the very profitable market of photocopiers. Just before 1970, the company embarked on an emerging discipline, that of information technology applied to the "office of the future" (by "office", we mean not only the work of secretaries, but also the administration of companies and workshops and research centers with their technical drawings), a discipline that would come to be called Office Automation.

At the end of the 1960s, we were still in the era of heavy computing, the computer market being held by IBM, Bull-Honeywell, Control Data, etc. Minicomputers were also beginning to appear (notably the PDP series from Digital Equipment Corporation), especially in the industrial world, and systems such as Unix (1971), primarily in the academic and research world. But, discreetly, a very different kind of computing was born, whose pioneers included Vannevar Bush, who had the first conception of hypertext; Douglas Engelbart, inventor of many of the modern concepts in the human–machine interface, including the mouse; without forgetting the American military, which funded significant research, including the ARPAnet, from which the Internet developed.

In 1970, Xerox created a research center in addition to its headquarters in Rochester (New York), with a strong focus on physics and chemistry, the Xerox Palo Alto Research Center (PARC; for its history, see [88] and [99]), located in California in what would later be called Silicon Valley. The mission of Xerox PARC is simple, at least in its statement: "Invent the office of the future."

### 9.1 Alto

Thanks to the proficiency of the researchers (many from Stanford and Berkeley), successes came very quickly: invention of the concept of the personal computer, and a prototype (Alto) with a screen and user interface with windows, icons, etc., manipulated by a mouse (it had been invented some years earlier, but this was the first use of it), invention of prototype printers, xerographic, chemical (like a modified Versatec) and laser; all were bitmapped, with resolutions between 300 and 400 dpi.

For the Alto, PARC developed a large variety of software, mostly office tools. These include: Bravo, the first WYSIWYG (What You See Is What You Get) text editor; drawing software including Draw, which allowed curves (actually cubic splines), hand drawings, and text elements to be integrated into a figure; Press, a "universal" (or portable, i.e. printer-independent) page description language (PDL), developed in 1975 for a complete description of documents (text and figures). This last would be followed by

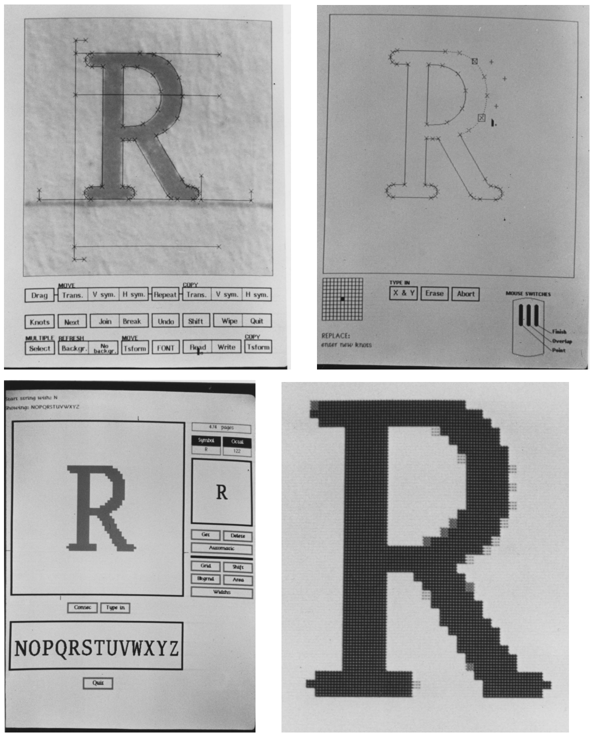


**Figure 66**: Screen images of the creation of fonts on the Alto (around 1975). With Fred: placing control points on a scan and improvements; with Prepress: a filled and improved bitmap (shades of gray indicate corrections, additions or deletions of pixels). [Courtesy Patrick Baudelaire [19] and Amelia Hugill-Fontanel (RIT Cary Graphic Art Collection)]

InterPress by Charles Geschke and John Warnock in 1980, and both would serve as a model for PostScript (see page 49).

## 9.2 The Alto font model

All Alto software used the same font model, developed by Bob Sproull and Patrick Baudelaire [20, 118]. The font creation system (figure 66) included an interactive spline editor and a filling program [19]. The first one, Fred, was written by Patrick Baudelaire (who had already designed the graphic language Draw, with procedures to draw splines). Fred projected onto a screen the image of a character to be digitized and, thanks to the mouse, one could draw the control points using B-splines (figure 66).

This Alto system was never commercialized, but was frequently presented and published in conferences and expert meetings and thus served as a catalyst. The Alto spawned many important successors, two of which are especially noteworthy:

- In 1978 Apple launched its personal computer project that would lead to the Macintosh (see page 33). In 1979, Steve Jobs bought from Xerox the right to exploit the research done at PARC, and Apple would benefit greatly from these revolutionary concepts.
- In 1982, Charles Geschke and John Warnock, two members of the PARC team (where they had developed InterPress based on Fred+Prepress), created Adobe, a company specialized in electronic publishing.

## 9.3 Birth of laser printers

Xerox was a leader in photocopying, and by 1972 had produced the first raster printer (XGP, the Xerox Graphics Printer, had a resolution of 192 dpi) to gain substantial use by computer scientists (at Carnegie-Mellon, Stanford, MIT, Caltech, and the University of Toronto).

Another PARC team worked on laser printers, led by Gary Starkweather. The first laser printer, EARS, was built and used with the Alto in 1973–76. In 1976 came the Dover printer, and in 1977 a color prototype. The first commercial product resulting from the work of Xerox PARC was the Xerox 9700 laser printer, inspired by the EARS prototype for the laser technology (at 300 dpi). It was distributed starting in 1977 and was a huge success worldwide.

IBM (which had been working on the problem of replacing impact printers since the 1960s) released an equivalent machine, the IBM 3800, in 1976. The Japanese company Canon also tackled the problem in the early 1970s and joined forces with Hewlett-Packard to produce a "big" laser printer, the HP 2680. It was not until 1983 that the first desktop printer, Canon's 300 dpi LBP-CX, was marketed by HP as the HP LaserJet. The same LBP-CX engine would be used in the Apple LaserWriter, the first commercial printer supporting PostScript.

## 10 Dissemination of the digital fonts concept

Through the end of the 1970s, research and development (technical and commercial) on digital typography was mainly carried out by small companies and university research laboratories. After 1980, thanks to laser printers, the concept of digital fonts spread at a very high speed and we have seen the birth and growth of foundries and development companies to the point where we are sometimes convinced that digital fonts were invented by Apple, Adobe, etc. It should be remembered that all the research, both earlier and today, could only be spread thanks to the community of researchers, scientific conferences and publications.

Typesetting with photocomposers was followed in particular by the ATypI conferences, but also revealed by experts like John W. Seybold, who created a consulting company for the graphic industries in 1963. In 1971, he launched, with his son Jonathan, the bi-monthly magazine *The Seybold Report* which remained for many years the canonical reference magazine for typesetting developments.

On the other hand, knowledge of digital fonts (apart from photocomposers) spread, initially, more in university circles and private research laboratories. This dissemination was made energetically and enthusiastically thanks to people interested and capable in two skills, at first sight mismatched: typography and data processing. The first attempts probably have been the Stanford classes, workshop and conferences organized at the end of the 1970s by Donald Knuth and his team around METAFONT (see page 42).

Let us quote too, in general: the ACM's Special Interest Group on Computer Graphics (SIGGRAPH), the magazine *Visible Language* (with articles by Vargo [100], Unger [122], and many others), the proceedings of the August 1983 conference of the ATypI at Stanford University [32, 31], the publication in 1983 of a "popular science" article by Bigelow (typographer) and Day (specialist in computer image processing) [28], the conferences *Electronic Publishing* and *Raster Imaging and Digital Typography* [109], and much more. All these communications are worthwhile above all because of the contacts that were made possible. Let us not forget also the publication of several books accessible by both communities, typographers and scientists, such as those by Seybold [114], Rubinstein [111], Alison Black [33] then Jorge De Buen [46], and Yannis Haralambous [57]. Finally, it is also worth mentioning magazines specialized in typography (*U&lc*, *Eye*, *PRINT*), etc. or not (like *TUGboat*!).

## 11 Adobe and PostScript

As we have seen, John Warnock and Charles Geschke worked at Xerox PARC and developed the InterPress page description language there. Because of the lack of interest by Xerox to commercially develop its revolutionary products, they left Xerox and founded Adobe in 1982.

There, Warnock and Geschke started to develop PostScript and looked for a desktop printer to market this language. For his part, Steve Jobs, at Apple, was looking to replace the ImageWriter (a dot matrix printer) of the first Macintosh, and discovered the LBP-CX Canon printer. Jobs then convinced Warnock to license PostScript to Apple for the LaserWriter, which would be marketed by Apple. A third partner then intervenes: Jonathan Seybold (son of John W. Seybold) had introduced to Apple another former PARC staff member, Paul Brainerd, who then founded the Aldus company, which developed PageMaker, and was another early PostScript licensee. It was the beginning of the success of PostScript and the success of Adobe.

In addition, Seybold's involvement had implications for the world of photocomposition — Aldus also entered this market. Linotype would be the first typesetter company to discover the importance of PostScript, and in 1984 released a PostScript raster image processor on its Linotronic 101 photocomposers, and then on the following model, the Linotronic 300. These photocomposers with 1270, 2540 and even 3300 dpi showed the very high quality of the PostScript fonts, which one could only imagine until then because of the relatively low 300 dpi resolution of the LaserWriter.

PostScript, conceived at the beginning for office automation, thus became the universal language (at least a portable language) in the world of pre-press and traditional printing.

Along with the commercial dissemination of PostScript — the customers being OEM (*Original Equipment Manufacturer*) companies in data processing and in particular the manufacturers of printers, photocomposers and computers — Adobe made efforts to spread knowledge of PostScript programming, which went far to expand its use. The best known is the publication of three manuals, respectively red (a reference manual), blue (a tutorial book of "recipes") and green (more oriented toward documents); see [7] to [10]. A fourth book (black, on the Type 1 format) would be published during the font wars.

We also find efforts to spread knowledge of PostScript in the booklets, sometimes luxurious, distributed by the branches like Adobe-France, in particular with font specimens.

### 11.1 The PostScript language

Based on graphic languages like Draw from PARC, and even more on InterPress, PostScript is a page markup language and not a document formatting language: it is up to the formatter to compose text, to hyphenate a word at the end of a line, etc. PostScript is designed to be the output language of typesetting programs; it is therefore analogous, in a general way, to the DVI language output by the original TeX.

Graphics supported in PostScript include line drawings, formed by line segments and Bézier curves (also supported in PARC's Fred software). Characters are only drawings, so in PostScript they become

```
/HH 100 def /H HH 2 div def
newpath
0 H moveto HH  0 rlineto 0 HH rlineto
HH neg 0 rlineto
closepath
.5 setgray fill % gray
HH HH moveto
H neg  H rlineto
H neg H 0 HH H H  rcurveto
H H HH 0 H H neg rcurveto closepath
1 0 0 setrgbcolor fill % red
0 setgray               % black
/Helvetica findfont 50 scalefont setfont
25 HH moveto (TYPO) show
```

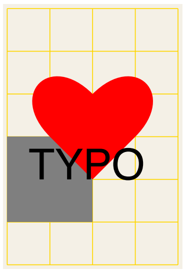


**Figure 67**: Example (typical around 1985) of PostScript program using Bézier curves (by the instruction `rcurveto`) and its result (the yellow grid, with a step of 50 points, is added).

procedures (routines) to draw them. A font is a dictionary of procedures for drawing characters, and their use is limited to two groups of operations: the selection of a font (with global metric properties and geometric properties depending on the graphic state) and the writing of a sequence of characters in this font. This simplicity of definition hides, however, a whole typographic machinery to which we will return.

**The PostScript machinery.** While PostScript is innovative in term of structures and possibilities, its most important feature is undoubtedly the way in which it functions. Until now, the drawing or font software computed bitmaps and sent them to the printer, but PostScript is more incremental [7]: the generation of bitmaps is done *inside* the printer; or, in front of the photocomposers, in a device called a *Raster Image Processor* (RIP).

Without going into details, we give two examples of PostScript programs. The first, figure 10, shows a letter 'P' with line segments used to approximate the curve of the bowl; we can observe that the instructions there resemble those of plotters and screens (figure 3). The other, figure 67, uses more concepts.

### 11.2 Type 3 and PostScript fonts

The font model defined using only the PostScript language [7] is what is now called *Type 3*. But in 1984, this number (3) only corresponded to a completely general method of generating characters, unlike the later-known Type 1 format, a less general method but additionally provided with hinting procedures.

**PostScript characters** are thus procedures (programs) describing their contours using line segments and Bézier curves. Their definition is independent of any particular size: PostScript uses geometric scaling, i.e. computes the coordinates scaled according to the desired body height, purely mathematically.

The PostScript font machinery is a special case of the general PostScript machinery, see [7] and [14]. It is important to remember that bitmaps are computed at the time of use; however, a caching mechanism allows these computations to be done only once per page for a given character, in a given font, at a given body height, etc. (unless this mechanism is disabled, see below).

**Glyphs and encoding tables.** Internally, a PostScript font defines its characters by their name: the character 'e' corresponds a PostScript procedure named `/e`. In fact, a PostScript font mainly consists of glyph-drawing procedures in the sense well-known today, but practically unknown then, in the 1980s. At the time of selecting a font, the PostScript interpreter builds an access table to at most 256 characters of the font, and an encoding vector to access this table, similar to character access via the `inputenc` package of LaTeX (in the years before Unicode). Also, just as TeX uses TFM (*TeX Font Metric*) information for typesetting, Adobe defined AFM (A for Adobe) metric information to accompany PostScript fonts.

With PostScript level 2, in 1991, it would become possible to call a character by its name (e.g. '`/ffi glyphshow`') even when it is not in the 256 character table. The list of these glyphs gradually became a standard for numeric fonts (via the so-called *glyphlist* file) and equivalences with Unicode names were made shortly before 2000.

**Dynamic fonts.** PostScript, like METAFONT, allows variables to be used in writing plot instructions, including characters. If these variables are given, for example, random values in METAFONT, since the bitmaps are calculated once when generating the fonts, before any final print, these values are not re-evaluated. A typical example is the Punk font [85] where each occurrence of 'E' is the same, though its form was generated randomly by METAFONT. That is, if we ran METAFONT again before typesetting the text, the 'E' would be different (figure 68, top).

By default, it is the same with PostScript since a cache mechanism avoids recalculating the bitmaps, for efficiency. However, with Type 3 fonts, the cache mechanism can be disabled. As a result, using a Type 3 version of Punk [103], we can get "real" random characters (before Beowolf by Erik van Blokland and Just van Rossum, for example) as shown in figure 68, bottom.

Beyond this playful aspect, this mechanism allows generating characters depending on the context (body, neighborhood, ...) or on, say, the time of day. However, few fonts have been developed in Type 3

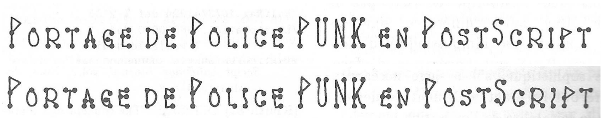

Portage de Police PUNK en PostScript
Portage de Police PUNK en PostScript

**Figure 68**: The same text written with Knuth's punk font: above with METAFONT [85], below in a PostScript Type 3 version [103]. In the first line, all E's are identical, which is not the case in the second.

format (although see [48, Type 3 Software]), and even fewer commercialized. The model was too general and required training to be able to interact with the font. Thirty years later, this concept has recently been rediscovered under the name of *variable fonts* and is very fashionable.

### 11.3 Type 1 fonts

Those who used PostScript at its beginning with the Type 3 font model with its cache mechanism thought that Adobe had hidden a "magic bullet" for fonts, and they were not wrong. Adobe kept more or less jealously secret a font model allowing faster and better rendering: Type 1. It was provided to certain foundries, under restrictive conditions, only after March 1985, and was not made public until 1990, with the publication of the "black book" at the time of the font wars [10].

Type 1 PostScript fonts work in essentially the same way as Type 3 fonts, but differ from Type 3 in that they are more professional or commercial by nature and, above all, by the possibility of programmable hinting.

**Professional aspects.** Although Adobe did not provide any typeface production software, third party software, such as Fontographer, saved designers from programming in raw PostScript. Type 1 fonts use the same Bézier curve contour description procedures as Type 3, and PostScript in general, but have a few other lower-level procedures that are particularly well suited to typography (vertical stems, horizontal lines, junctions, etc.) and are much more efficient than the general *lineto* and *curveto* operators of Type 3.

Type 1 fonts also offer a series of controls related to font secrecy (copyright, identification number) etc., and elaborate encryption methods! The encryption algorithm was disclosed by Adobe in the Type 1 book.

**Hinting.** More important typographically, hinting instructions in Type 1 allow giving instructions to the bitmap rendering procedures to improve the final drawing of the characters (see earlier discussion and examples, page 41). We have seen that Ikarus proposed them as early as 1983 (figure 56). Earlier,

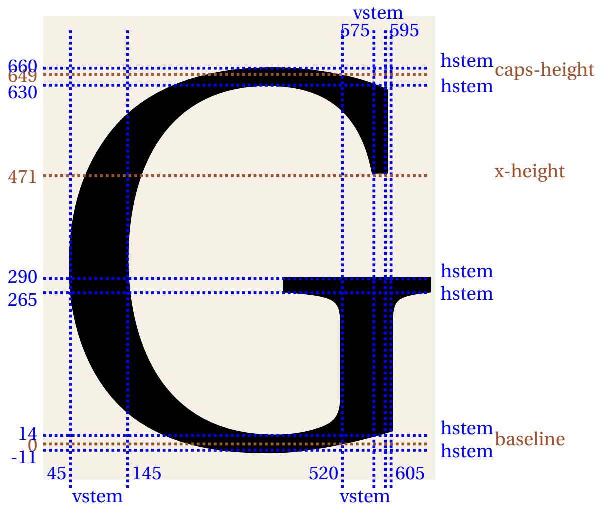


**Figure 69**: Hinting in Type 1 PostScript fonts is done through "blue lines". Here they define various horizontal (`hstem`) and vertical (`vstem`) thicknesses. Following [10].

Fred and METAFONT did not need explicit hinting because they generated the bitmaps themselves.

The Type 1 hinting operators are complicated. and are intended primarily to help render fonts at relatively high (printer) resolutions, not for screens. The method chosen by Adobe was that of so-called "blue lines", entered by the designer to specify constraints (figure 69).

### 11.4 The Adobe font library

As mentioned above, Adobe did not initially offer any tools for writing fonts. Instead, Adobe entered, if not revolutionized, the font market by digitizing a huge number of fonts and implementing them in PostScript. According to Peter Karow [71, p. 269], the first 250 outline-based fonts distributed by Adobe were purchased from URW.

## 12 As a matter of conclusion

With the advent of PostScript and laser printers, the prehistory of the digital fonts ends. Let us say simply in a few words what happened next ...

- The widespread adoption of PostScript with its cubic Bézier curves, including for printing, had immediate consequences for other software in the field, which adapted to the current tastes. For example, TeX quickly supported PostScript with a new DVI conversion program, `dvips`; METAFONT itself was not changed, but a companion program METAPOST [64] was developed to output encapsulated PostScript instead of bitmaps;

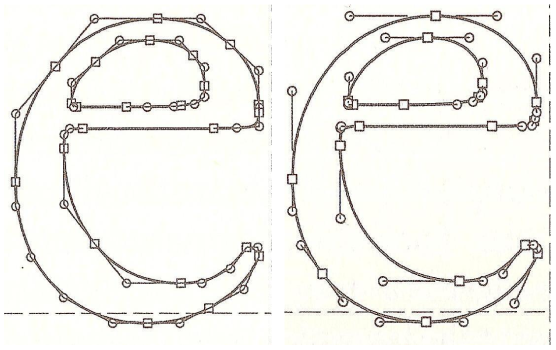


**Figure 70**: The same 'e' implemented, on the left, in TrueType (quadratic splines) and, on the right, in Type 1 (cubic splines). After Bringhurst [40, p. 184].

in addition, programs such as Metafog were developed to extract the curves from inside METAFONT and output Type 1 or Type 3 fonts. Troff also had PostScript output early. Ikarus kept its IK format but made a BE version where the original conics are translated into cubic Béziers (the translation is straightforward).

Other conic spline formats were tried. Let us mention in particular the F3 format of Vaughan Pratt for Sun [106, pp. 144, 331].

- But Adobe kept to itself the rights to the Type 1 format and machinery, so other manufacturers, notably Apple, later joined by Microsoft, developed an alternative. This led to the birth of the TrueType format and to the release by Adobe to the public of the Type 1 format, in 1989. Each model, Type 1 and TrueType, has its own partisans defending the superiority of their hints or their particular splines, according to the needs of type designers (figure 70 has a comparison).
- The various font formats and new encodings, in particular Unicode, provided for portability of fonts, and their use for languages with different character sets.
- In the late 1970s, screens were developed with pixels that are not black or white, but grayscale. At the end of the 1990s, liquid crystal displays appeared where pixels could be divided into subpixels. In the 2000s, three sub-pixel renderers were in use: Adobe's CoolType, Apple's ATSUI (*Type Services for Unicode Imaging*, using the Quartz2D engine), and Microsoft's ClearType. All have been significantly enhanced and/or replaced in the years since.
- A whole collection of font creation systems were developed (FontForge, Fontlab, Fontographer, FontStudio, Glyphs, etc.). Many of them were developed by individuals (such as Von Ehr, Yuri Yarmola, and George Williams) before being taken over by larger companies.
- The most important point of this period is what is called "the font wars" opposing TrueType and Type 1 supporters (see Bigelow [27]). Although the background was technical (choice of conics or cubics and especially method of hinting), it played out mainly with commercial connotations. TrueType and Type 1 eventually converged with OpenType (1993).
- At the time of the original "movable types", a cast was a set of classified types, possibly with the mechanical composition (e.g., the Monotype machines). With the second generation phototypesetters we see appear side information: the width tables. These then also existed for digital Hershey fonts. In the Ikarus formats, TeX and METAFONT, and PostScript fonts accompanied these width tables with information on ligatures and kerning (TFM and AFM respectively). OpenType has taken over and considerably increased this mechanism of side tables (gpos, etc.). Its strength is thus based on the experience of all the preceding work.
- It seems to us personally that the evolution of the future fonts will be based on the side tables by increasing their content (in particular by the use of variables) but also by using these tables not just at the time of their loading, but also during composition (these will then be real variable fonts).

Finally, we would like to point out that all the tools (except tentatively METAFONT) that we have shown are more manufacturing tools — the drawing of a character that already exists, even if only in the form of a sketch, than creation (from scratch). As yet, we have no answer to the philosophical questions of Douglas Hofstadter (What is the essence of a-ness?) or Richard Southall (Are the shape and the appearance of a character identical?).

By way of final words, I'd like to conclude with an homage to Southall by quoting these words by Gerry Leonidas [89]:

> Richard's ideas about "models" and "patterns" in type design are the definitive starting point for any discussion of typemaking, and — with some adjustments for terminology — absolutely essential in any review of typeface design processes with digital tools. In fact, the growth of rendered instances of typeforms across many devices make his ideas more relevant than ever, and prove that his approach

provides the key ideas for discussing typeface design across type-making technologies. Together with some texts by Robin Kinross, his writings [for a list, see [16]], are amongst the very few indispensable texts for any theoretical discussion of typeface design.

**Acknowledgments**

The author would like to thank the many people who provided illustrations for this article, and also the people who helped him in the drafting of this text and then its revision, in particular Patrick Baudelaire, Charles Bigelow, Yannis Haralambous, Roger Hersch, Vania Joloboff, Peter Karow, Christian Laucou, and Alan Marshall.

For this publication in *TUGboat*, he would also like to thank Patrick Bideault for his English translation, and Barbara Beeton and Karl Berry for their deep proofreading and remarkable work as editors.

## References

[1] *Histoire de l'écriture typographique – De Gutenberg au XVII^e siècle*, by Yves PERROUSSEAUX. Atelier Perrousseaux éd./Adverbum, 2004.

[2] *Histoire de l'écriture typographique – Le XVIII^e siècle*, tome I/II, by Yves PERROUSSEAUX. Atelier Perrousseaux éd./Adverbum, 2010.

[3] *Histoire de l'écriture typographique – Le XVIII^e siècle*, tome II/II, by Yves PERROUSSEAUX. Atelier Perrousseaux éd./Adverbum, 2010.

[4] *Histoire de l'écriture typographique – Le XIX^e siècle français*, by Jacques ANDRÉ and Christian LAUCOU. Atelier Perrousseaux éd./Adverbum, 2013.

[5] *Histoire de l'écriture typographique – Le XX^e siècle, de 1900 à 1950*, collective work under the direction of Jacques ANDRÉ. Atelier Perrousseaux éd./Adverbum, 2016.

[6] *Histoire de l'écriture typographique – Le XX^e siècle, de 1950 à 2000*, collective work under the direction of Jacques ANDRÉ. Atelier Perrousseaux éd./Adverbum, 2016.

[7] Adobe Systems Inc., *PostScript Language Reference Manual*, first edition, Reading, MA: Addison-Wesley, 1985; second edition, 1991.

[8] Adobe Systems Inc., *PostScript Language Tutorial and Cookbook*, Reading, MA: Addison-Wesley, 1985.

[9] Adobe Systems Inc., *PostScript Language Program Design*, Reading, MA: Addison-Wesley, 1988.

[10] Adobe Systems Inc., *The Type 1 Format Specification*, Reading, MA: Addison-Wesley, 1990.

[11] Jacques ANDRÉ, *Création de fontes en typographie numérique*, Mémoire d'HDR, Université Rennes I, 29 sept. 1993, 124 pp. `theses.hal.science/tel-00011218/file/andre.pdf`

[12] Jacques ANDRÉ, *Courier – Histoire d'un caractère – De la machine à écrire aux fontes numériques*, éd. du Jobet, 1993. `jacques-andre.fr/fontex/courier.pdf`

[13] Jacques ANDRÉ, De Pacioli à Truchet : trois siècles de géométrie pour les caractères, *4 000 ans d'histoire des mathématiques : les mathématiques dans la longue durée – 13^e colloque Inter-IREM d'épistémologie et histoire des mathématiques*, IREM-Rennes, mai 2000, pp. 1–38. `hal.inria.fr/inria-00000956`

[14] Jacques ANDRÉ and Justin BUR, Métrique des fontes PostScript, *Cahiers Gutenberg*, n° 8 (1991), pp. 29–50. `http://numdam.org/item/CG_1991___8_29_0/`

[15] Jacques ANDRÉ and Denis GIROU, Father Truchet, the typographic point, the Romain du roi, and tilings, *TUGboat*, Vol. 20 (1999), No. 1, pp. 8–14. `tug.org/TUGboat/tb20-1/tb62andr.pdf`

[16] Jacques ANDRÉ and Alan MARSHALL, Richard Southall: 1937–2015, *TUGboat*, Vol. 36 (2015), No. 2, pp. 100–102. `tug.org/tugboat/tb36-2/tb113southall.pdf`

[17] Jacques ANDRÉ and Moncef MLOUKA (eds.), *Workshop on Font Design Systems*, INRIA-Sophia, May 1987. See also [109], 1989.

[18] AUGUSTIN, *Wim Crouwel*, Index Grafik, 7 avril 2014. `http://indexgrafik.fr/wim-crouwel/`

[19] Patrick BAUDELAIRE, *The Fred User's Manual*, Internal Report, Xerox Palo Alto Research Center, Palo Alto, California, 1976.

[20] Patrick BAUDELAIRE, The Xerox Alto Font Design System, in [31].

[21] Patrick BAUDELAIRE and M. STONE, Techniques for Interactive Raster Graphics, SIGGRAPH 80 Proceedings, *Computer Graphics*, Vol. 14, No. 3, 1980.

[22] Barbara BEETON, Karl BERRY and David WALDEN, TEX: A Branch in Desktop Publishing Evolution, Part 1, *IEEE Annals of the History of Computing*, Vol. 40, No. 3, Jul./Sept. 2018, pp. 78–93. `ieeexplore.ieee.org/document/8509554/`

[23] Yves BEKKERS, Daniel HERMAN, and Michel RAYNAL, *Conception et réalisation d'une machine-langage de haut niveau adaptée à l'écriture de systèmes*, Ph.D. thesis, Rennes University, 24 sept. 1975.

[24] Charles BIGELOW, Les caractères rationalisés, in *La manipulation de documents* (Jacques André, ed.), INRIA-Centre de Rennes, mai 1983, pp. 15-27. `jacques-andre.fr/japublis/manip83.pdf`

[25] Charles Bigelow, Review and Summaries of *The History of Typographic Writing — The 20th century*. Originally published in three parts in *TUGboat* Vol. 38 (2017); combined: tug.org/books/reviews/tv38bigelow.pdf

[26] Charles Bigelow, Typeface Features and Legibility Research, *Vision Research*, Vol. 165, Dec. 2019, pp. 162–172. doi.org/10.1016/j.visres.2019.05.003

[27] Charles Bigelow, The Font Wars, Parts 1 and 2, *IEEE Annals of the History of Computing* (Special issue: History of Desktop Publishing), Vol. 42, No. 1, Jan./Mar. 2020, pp. 7–40. doi.org/10.1109/MAHC.2020.2971202 doi.org/10.1109/MAHC.2020.2971745

[28] Charles Bigelow and Donald Day, Digital Typography, *Scientific American*, Vol. 249, No. 2, pp. 106–119, Aug. 1983.

[29] Charles Bigelow and Kris Holmes, Notes on Apple 4 Fonts, *Electronic Publishing*, Vol. 4, No. 3, Sept. 1991, pp. 171–181. http://cajun.cs.nott.ac.uk/compsci/epo/papers/volume4/issue3/ep050cb.pdf

[30] Charles Bigelow and Kris Holmes, The Design of Lucida: An Integrated Family of Types for Electronic Literacy, in *Text Processing and Document Manipulation* (J.C. van Vliet, ed.), Cambridge University Press, 1986.

[31] Charles Bigelow and Kevin Larson (eds.), *Visible Language* (Special issue: Reflecting on 50 Years of Typography, Vol. 50, No. 2, Aug. 2016. journals.uc.edu/index.php/vl/issue/view/461

[32] Charles Bigelow and Lynn Ruggles (eds.), *Visible Language* (Special issue: The Computer and the Hand in Type Design), Vol. 19, No. 1, Winter 1985. journals.uc.edu/index.php/vl/issue/view/369

[33] Alison Black, *Typefaces for Desktop Publishing: A User Guide*, London: Architecture Design and Technology Press, 1990.

[34] Lewis Blackwell, *20th-Century Types*, Lawrence King Publishing, 2004.

[35] Gérard Blanchard (coordinated by), *L'écriture télématique, années zéro*, Les Cahiers de Lure, 1985.

[36] Gérard Blanchard, *L'eredita Gutenberg*, Gianfranco Altieri Editore, 1989.

[37] Gérard Blanchard, *Aide au choix de la typo-graphie – Cours supérieur*, Atelier Perrousseaux éd., 1998.

[38] Paul Bourke, *Hershey Vector Font based on the Hershey character set*, Oct. 1977. http://paulbourke.net/dataformats/hershey/

[39] Jack E. Bresenham, Algorithm for computer control of a digital plotter, *IBM Systems Journal*, Vol. 4, No. 1, Jan. 1965, pp. 25–30. doi.org/10.1147/sj.41.0025

[40] Robert Bringhurst, *The Elements of Typographic Style*, Hartley & Marks publishers, 4th edition, 2015.

[41] *CalComp Software Reference Manual*, California Computer Products Inc., Oct. 1976. archive.org/details/bitsavers_calcompCalceManualOct76_6872751

[42] Edward M. Catich, *The Origin of the Serif: Brush Writing and Roman Letters*, Davenport, IA: The Catfish Press, 1968.

[43] Philippe J.M. Coueignoux, *Generation of Roman Printed Fonts*, Ph.D. thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 1975. dspace.mit.edu/bitstream/handle/1721.1/27408/02149218-MIT.pdf

[44] Dave Crossland, “Why didn't METAFONT catch on?”, *TUGboat*, Vol. 29 (2008), No. 3, pp. 418-420. tug.org/TUGboat/tb29-3/tb93crossland.pdf

[45] *Typographic Architectures typographiques*, texts by Wim Hendrik Crouwel, Catherine de Smet and Emmanuel Bérard, Editions fsept F7, Paris, 2007.

[46] Jorge De Buen, *Manual de diseño editorial*, Santilano, Mexico, 2000,

[47] Christian Delorme and Jacques André, Le Delorme, un caractère modulaire et dépendant du contexte, *Communication et langages*, Vol. 86 (1990), pp. 64–76. www.persee.fr/web/revues/home/prescript/article/colan_0336-1500_1990_num_86_1_2261

[48] Jean-Luc Devroye, *Type Design, Typography, Typefaces and Fonts: An encyclopedic treatment of type design, typefaces and fonts.* Web page closed on May 6, 2022. http://luc.devroye.org/fonts.html

[49] Jean-Luc Devroye, METAFONT links, in [48]. http://luc.devroye.org/metafont.html

[50] Diderot & D'Alembert, *Encyclopédie, ou Dictionnaire Raisonné des Sciences, des Arts et des Métiers*, 1751–1772. encyclopedie.uchicago.edu

[51] Albrecht Dürer, Of the just shaping of letters. www.zigzaganimal.be/elements/just_shaping_scan.pdf

[52] Jean-Jacques Eltgen, Techniques d'impression d'images numérisées, *Techniques de l'ingénieur*, art. E-5-670, 1992.

[53] James Essinger, *Jacquard's Web: How a hand loom led to the birth of the information age.* Oxford, U.K.: Oxford University Press, 2004.

[54] Bernard Ficatier and Hugues Roche, *Concevoir, relever et dessiner des plans de voiliers classiques et traditionnels*, Douarnenez: Chasse-marée, 2004.

[55] Richard Furuta, Jeffrey Scofield, and Alan Shaw, Document Formatting Systems: Survey, Concepts, and Issues, *Computing Surveys*, Vol. 14, No. 3, Sept. 1982, pp. 417–472. `doi.org/10.1145/356887.356891`

[56] Pierre-Marie Gallois, *Quand Paris Était Ville-Lumière*, L'Âge D'homme, 2001.

[57] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.

[58] Tamir Hassan, Changyuan Hu, and Roger D. Hersch, Next Generation Typeface Representations: Revisiting Parametric Fonts, *ACM DocEng 2010 conference*, Sept. 2010, pp. 181–184. `lspwww.epfl.ch/publications/typography/ngtrrpf_10.pdf`

[59] Rudolf Hell, Le Digiset, composeuse binaire électronique, *Caractère*, vol. 16, n° 11, 1965, pp. 5–16.

[60] Roger Hersch (ed.), *The Visual and Technical Aspects of Type*, Cambridge University Press, 1993. `lspwww.epfl.ch/publications/typography/vataot.html`

[61] Allen V. Hershey, *Calligraphy for Computers*, U.S. Naval Weapons Laboratory, 1967, 500pp. `archive.org/details/hershey-calligraphy_for_computers`

[62] Allen V. Hershey, A Computer System for Scientific Typography, *Computer Graphics and Image Processing*, Vol. 1 (1972), pp. 273–385.

[63] John D. Hobby, *Digitized Brush Trajectories*, Ph.D. thesis, Stanford University, Aug. 1985. `tug.org/docs/hobby/hobby-thesis.pdf`

[64] John D. Hobby, *A User's Manual for METAPOST*. AT&T Bell Laboratories Computing Science Technical Report 162, 1992. With updates: `tug.org/docs/metapost/mpman.pdf`

[65] Douglas Hofstadter, *Metamagical Themas*, Basic Books, 1985. `archive.org/details/MetamagicalThemas`

[66] Kris Holmes, *Dossier — Calligraphy, Lettering, Signage and Graphic Design, Filmmaking and Articles*, Keepsake for the Frederic Goudy Award, Rochester Institute of Technology, 2012.

[67] Changyuan Hu and Roger D. Hersch, Parameterizable Fonts Based on Shape Components, *IEEE Computer Graphics and Applications*, Vol. 21, No. 3, May/June 2001, pp. 70–85. `lspwww.epfl.ch/publications/typography/pfbosc.pdf`

[68] Peter Karow, *Digital Formats for Typefaces*, URW Verlag, Hamburg, 1987. `archive.org/details/digitalformatsfo0000karo`

[69] Peter Karow, Digital punch cutting, *Electronic Publishing*, Vol. 4, No. 3, Sept. 1991, pp. 151–170. `http://cajun.cs.nott.ac.uk/compsci/epo/papers/volume4/issue3/ep044pk.pdf`

[70] Peter Karow, *Digital Typefaces: Description and Fprmats*, Springer-Verlag, 1994. `books.google.com/books?id=oomrCAAAQBAJ`

[71] Peter Karow, Two decades of typographic research at URW: A retrospective, in [109, pp. 265–304 (1998)].

[72] Peter Karow, *Font Technology: Methods and Tools*, Springer Science, 2012.

[73] Peter Karow, Digital Typography & Artificial Intelligence, On the occasion of the presentation of the third *Dr. Peter Karow Award for Font Technology & Digital Typography* to Dr. Donald E. Knuth at the ATypI Amsterdam 2013 conference, Adobe and Dutch Type Library, 2013.

[74] Brian W. Kernighan, *A Typesetter-independent TROFF*, Computing Science Technical Report No. 97, Bell Labs, revised, Mar. 1982. `archive.org/details/typesetter-independent-troff`

[75] Brian W. Kernighan, PIC — a language for typesetting graphics, *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation*, June 1981, pp. 92–98. `doi.org/10.1145/800209.806459` Revised publication, May 1991: `archive.org/details/pic-graphics-language`

[76] Brian W. Kernighan and Lorinda L. Cherry, A System for Typesetting Mathematics, *Communications of the ACM*, Vol. 18, No. 3, Mar. 1975, pp. 151–157. `dl.acm.org/doi/10.1145/360680.360684`

[77] Christopher Knoth, *Computed Type Design*, Master Art Direction, ECAL Lausanne, 2011. `christoph-knoth.com/shared/computed_type_-_christoph_knoth.pdf`

[78] Donald E. Knuth, "TAU EPSILON CHI: A System for Technical Text", STAN-CS-78-675.1, Computer Science Department, Stanford University, Stanford, California, Nov. 1978. `purl.stanford.edu/jy605yq4819`

[79] Donald E. Knuth, Mathematical Typography, *Bulletin (N.S.) of the American Mathematical Society*, Vol. 1, No. 2, 1979, pp. 337–372. `doi.org/10.1090/S0273-0979-1979-14598-1`

[80] Donald E. Knuth, *TEX and Metafont — New directions in typesetting*, Digital Press and American Mathematical Society, 1979.

[81] Donald E. Knuth, The Letter S, *The Mathematical Intelligencer*, Vol. 2 (1980), pp. 114–122.

[82] Donald E. Knuth, The Concept of a Meta-Font, *Visible Language*, Vol. 16, No. 1, Jan. 1982, pp. 3-27. `journals.uc.edu/index.php/vl/article/view/5329`

[83] Donald E. Knuth, *The METAFONTbook*, Computers & Typesetting, Reading, MA: Addison-Wesley, 1986.

[84] Donald E. KNUTH, *Computer Modern Typefaces*, Reading, MA: Addison-Wesley, 1986.

[85] Donald E. KNUTH, A Punk Meta-Font, *TUGboat*, Vol. 9 (1988), No. 2, pp. 152–168. `tug.org/TUGboat/tb09-2/tb21knut.pdf`

[86] Donald E. KNUTH, *Digital Typography*, xvi+685pp. CSLI Lecture Notes, no. 78, Stanford, California, 1999.

[87] Eliyezer KOHEN, A simple and efficient way to design middle resolution fonts, in [17, pp. 3–19] and [109, pp. 22–33 (1989)].

[88] Sacha KRAKOWIAK, Xerox PARC et la naissance de l'informatique contemporaine, *Interstices* (revue Inria en ligne), 2012. `interstices.info/jcms/int_64091/xerox-parc-et-la-naissance-de-l-informatique-contemporaine`

[89] Gerry LEONIDAS, Farewell, Richard Southall, 17 June 2015. `leonidas.net/2015/06/17/farewell-richard-southall/`

[90] Raph LEVIEN and Carlo H. SÉQUIN, Interpolating Splines: Which is the fairest of them all?, *Computer-Aided Design & Applications*, Vol. 6, No. 1, 2009, pp. 91–102. `http://graphics.berkeley.edu/papers/Levien-IIS-2009-06/`

[91] Pierre MACKAY, The KATIB System, a revolutionary advancement in Arabic script typesetting by means of the computer, *Scholarly Publishing* Vol. 8, No. 2, 1977, pp. 142–150.

[92] Pierre A. MACKAY, Looking at the Pixels. Quality Control for 300 dpi Laser Printer Fonts, Especially METAFONTs, in [109, pp. 205–217 (1991)].

[93] Julien MAILLAND and Kevin DRISCOLL, *Minitel: The Online World France Built Before the Web*, 20 June 2017. `spectrum.ieee.org/minitel-the-online-world-france-built-before-the-web`

[94] Ladislas MANDEL, Un caractère pour annuaires téléphoniques, *Communication et langages*, n° 39, 1979, pp. 51–61.

[95] Ladislas MANDEL, Naissance d'une écriture – Réflexions sur la typographie et la télématique, dans *L'écriture télématique, années zéro* [35, pp. 41–49].

[96] Ladislas MANDEL, *Du pouvoir de l'écriture*, Atelier Perrousseaux éd., 1998.

[97] Marie MARCHAND, *La Grande Aventure du Minitel*, Librairie Larousse, 1987.

[98] M. V. MATHEWS, Carol LOCHBAUM, and Judith A. MOSS, Array: Three Fonts of Computer-drawn Letters, *The Journal of Typographic Research*, Vol. 1, No. 4, Oct. 1967, pp. 345–356. `journals.uc.edu/index.php/vl/article/view/5008`

[99] Paul MCJONES, Xerox Alto file system archive, Computer History Museum, last revised 9 Nov. 2017. `xeroxalto.computerhistory.org`

[100] H.W. MERGLER and P.M. VARGO, One Approach to Computer Assisted Letter Design, *The Journal of Typographic Research*, Vol. 2, No. 4, Oct. 1968, pp. 299–322. `journals.uc.edu/index.php/vl/article/view/5032`

[101] Stanley MORISON, On Some Italian Scripts of the XV and XVI Centuries, in *Letter forms, typographic and scriptorial: Two essays on their classification, history, and bibliography*, Typophiles, pp. 95–129, 1968.

[102] Heidrun OSTERER and Philipp STAMM, *Adrian Frutiger – Caractères: L'œuvre Complète*, Walter de Gruyter, Switzerland, 2012.

[103] Victor OSTROMOUKHOV and Jacques ANDRÉ, Punk : de METAFONT à PostScript, *Cahiers GUTenberg*, n° 4 (1989), pp. 23–28. `http://numdam.org/item/CG_1989___4_23_0/`

[104] Scott PAKIN, *The Comprehensive LaTeX Symbol List*, 2021. `ctan.org/pkg/comprehensive`

[105] Arthur PHILLIPS, *Computer Peripherals & Typesetting*, London, Her Majesty's Stationery Office, 1968.

[106] Vaughan PRATT, Techniques for conic splines, *ACM SIGGRAPH Computer Graphics*, Vol. 19, No. 3, July 1985, pp. 151–159. `doi.org/10.1145/325165.325225`

[107] Lilian M.C. RANDALL, A Nineteenth Century 'Medieval' Prayerbook Woven in Lyon, in *Art the Ape of Nature: Studies in Honor of H.W. Janson*, Moshe Barasch, Lucy F. Sandler (eds), New York, NY: Harry N. Abrams, 1981, pp. 651–668.

[108] Brian K. REID and David HANSON, An annotated bibliography of background material on text manipulation, *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation*, ACM SIGPLAN Notices, Vol. 16, No. 6, June 1981, pp. 157–160. `doi.org/10.1145/800209.806467`

[109] RIDT, conference series proceedings:
- *Raster Imaging and Digital Typography*, Lausanne, Oct. 1989 (Jacques ANDRÉ and Roger HERSCH, eds.), Cambridge University Press, 1989. `books.google.com/books?id=mjO9AAAAIAAJ`
- *Raster Imaging and Digital Typography II*, Boston, Oct. 1991 (Robert A. MORRIS and Jacques ANDRÉ, eds.), Cambridge University Press, 1991. `books.google.com/books?id=Q9KtGcpfNgUC`
- *Raster Imaging and Digital Typography*, special issue of *Electronic Publishing Origination Dissemination and Design*, (Jacques ANDRÉ, Jakob GONCZAROWSKI, and Richard SOUTHALL, eds.), Wiley, 1994. `books.google.com/books?id=gJcVAQAAIAAJ`
- *Electronic Publishing, Artistic Imaging, and Digital Typography* (Roger HERSCH,

Jacques André, and Heather Brown, eds.), Lecture Notes in Computer Science #1375, Springer-Verlag, 1998. books.google.com/books?id=bo453EDNBp4C

[110] Frank Romano (with Miranda Mitrano), *History of Desktop Publishing*, Oak Knoll Press, 2019.

[111] Richard Rubinstein, *Digital Typography — an introduction to type and composition for computer system design*, Reading, MA: Addison-Wesley, 1988.

[112] Lynn Ruggles, *Letterform Design Systems*, Stanford University Technical Report STAN-CS-83-971, 1973. http://i.stanford.edu/pub/cstr/reports/cs/tr/83/971/CS-TR-83-971.pdf

[113] Stewart C. Russell, *Hershey Font Outlines*, May 2014. scruss.com/wordpress/wp-content/uploads/2014/05/hershey_samples.pdf

[114] John Seybold and Fritz Dressler, *Publishing From the Desktop*, New York, NY: Bantam Books, 1987. archive.org/details/publishingfromde0000seyb

[115] Richard Southall, Interfaces between the designer and the document, in *Structured documents* (Jacques André, Richard Furuta, and Vincent Quint, eds.), Cambridge University Press, 1989, pp. 119–131. dl.acm.org/doi/10.5555/73173.73179

[116] Richard Southall, METAFONT in the Rockies: The Colorado Typemaking Project, in *EP'98* [109, 167–180 (1998)]; link.springer.com/chapter/10.1007%2FBFb0053270. Republished in *Computers and Typography 2* (Rosemary Sassoon, ed.), Intellect Books, 2002; books.google.com?id=wdYmvQD5C8IC

[117] Richard Southall, *Printer's Type in the Twentieth Century — Manufacturing and Design Methods*, The British Library/Oak Knoll Press, 2005.

[118] Bob Sproull, *Font Representations and Formats*, Internal note, Xerox PARC, Mar. 1977. xeroxparcarchive.computerhistory.org/indigo/printingdocs/.FONTFORMATS.PRESS!1.pdf

[119] David R. Siegel, *The Euler Project at Stanford*, The Department of Computer Science, Stanford University, Stanford, 1985.

[120] David Sudweeks, Type Trends: Superelliptical Type, *FontShop Typographic Trends*, Nov. 2012. fontshopblog.wordpress.com/2012/11/22/type-trends-superelliptical-type

[121] Edward Tufte, *Visual Explanations — Images and Quantities, Evidence and Narrative*, Cheshire, CT: Graphic Press, 1997.

[122] Gerard Unger, The Design of a Typeface, *Visible Language*, Vol. 13, No. 2, Apr. 1979, pp. 134–149. journals.uc.edu/index.php/vl/article/view/5266

[123] Gerard Unger, *in* Other Replies to Donald E. Knuth's article "The Concept of a Meta-Font", *Visible Language*, Vol. 16, No. 4, Oct. 1982, pp. 353–356. journals.uc.edu/index.php/vl/issue/view/360

[124] Andries Van Dam and Eric E. Rice, On-line Text Editing: A Survey, *Computing Surveys*, Vol. 3, No. 3, Sept. 1971, pp. 93–114. doi.org/10.1145/356589.356591

[125] Yue Wang, Interview with Charles Bigelow, *TUGboat*, Vol. 34 (2013), No. 2, pp. 136–167. tug.org/TUGboat/tb34-2/tb107bigelow-wang.pdf

[126] Matthew Westerby, *The Woven Prayer Book: Cocoon to Codex*, Satellite Series. Paris, France & Chicago, IL, USA: Les Enluminures, 2019.

[127] Norman M. Wolcott and Joseph Hilsenrath, *A Contribution to Computer Typesetting Techniques: Tables of coordinates for Hershey's Repertory of Occidental Type Fonts and Graphic Symbols*, National Bureau of Standards, NBS Special Publication 424, Apr. 1976. scruss.com/wordpress/wp-content/uploads/2014/04/tables_of_coordinates_for_hersheys_repertory_of_occidental_type_fonts-wolcott_and_hilsenrath.pdf

[128] Norman M. Wolcott, FORTRAN IV Enhanced Character Graphics, National Bureau of Standards, Institute for Computer Sciences and Technology, NBS Special Publication 500-32, Apr. 1978, 64 pp. archive.org/details/fortranivenhance5003wolc

[129] Hermann Zapf, *Hermann Zapf and His Design Philosophy*, Chicago, IL: Society of Typographic Arts, 1987. Introduction by Carl Zahn.

[130] Herman Zapf, Vom Formgesetz der Renaissance-Antiqua, *Der Polygraph*, Heft 21.

◇ Jacques André
https://jacques-andre.fr

## Typographers' Inn

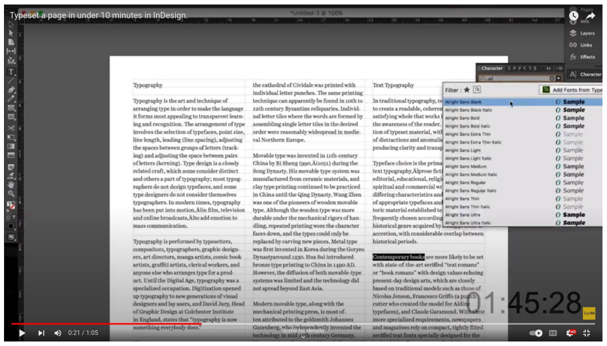
Peter Flynn

### Fast startup with LaTeX — a video challenge

It's usually unfair to compare LaTeX with other typesetting systems such as InDesign or FrameMaker, and the old DTP favourites like PageMaker, Ventura, 3B2, and QuarkXPress, some of which are still in use. The reason is not just in some way that 'LaTeX is "better"' — whether you believe that to be true or not — it's that they *work differently*, and LaTeX has a level of programmability and style of automation not as easily accessible in other systems.

Other systems do of course have macros and styles, and many have an internal scripting language. InDesign uses Extendscript;[1] XPress uses AppleScript; Frame uses Extendscript also; PageMaker uses VBScript; 3B2 offers its own scripting language as well as Perl. But LaTeX's programmability is not an add-on or plugin or third-party language, it *is* the TeX typesetting language, inherently a part of the design from the ground up; and LuaLaTeX provides a language with external acceptance.

So when my attention was drawn to a YouTube video put out by Los Angeles online design educators Type-Ed, called *Typeset a page in under 10 minutes in InDesign* [2], I wondered how that would compare to LaTeX. We're not talking here about including the learning curve in either case — that would stretch it to an hour or so for LaTeX and I have no idea how long it takes to learn InDesign to the level needed. There is no voice-over, just music and clicks.



Typography

Typography is the art and technique of arranging type in order to make the language it forms most appealing to transparent learning and recognition. The arrangement of type involves the selection of typefaces, point size, line length, leading (line spacing), adjusting the spaces between groups of letters (tracking) and adjusting the space between pairs of letters (kerning). Type design is a closely related craft, which some consider distinct and others a part of typography; most typographers do not design typefaces, and some type designers do not consider themselves typographers. In modern times, typography has been put into motion,Äìin film, television and online broadcasts,Äìto add emotion to mass communication.

Typography is performed by typesetters, compositors, typographers, graphic designers, art directors, manga artists, comic book artists, graffiti artists, clerical workers, and anyone else who arranges type for a product. Until the Digital Age, typography was a specialized occupation. Digitization opened up typography to new generations of visual designers and lay users, and David Jury, Head of Graphic Design at Colchester Institute in England, states that "typography is now something everybody does."

the cathedral of Cividale was printed with individual letter punches. The same printing technique can apparently be found in 10th to 12th century Byzantine reliquaries. Individual letter tiles where the words are formed by assembling single letter tiles in the desired order were reasonably widespread in medieval Northern Europe.

Movable type was invented in 11th-century China by Bi Sheng (990,Äì1051) during the Song Dynasty. His movable type system was manufactured from ceramic materials, and clay type printing continued to be practiced in China until the Qing Dynasty. Wang Zhen was one of the pioneers of wooden movable type. Although the wooden type was more durable under the mechanical rigors of handling, repeated printing wore the character faces down, and the types could only be replaced by carving new pieces. Metal type was first invented in Korea during the Goryeo Dynastyaround 1230. Hua Sui introduced bronze type printing to China in 1490 AD. However, the diffusion of both movable-type systems was limited and the technology did not spread beyond East Asia.

Modern movable type, along with the mechanical printing press, is most often attributed to the goldsmith Johannes Gutenberg, who independently invented the technology in mid-15th century Germany.

Text Typography

In traditional typography, te
to create a readable, coheren
satisfying whole that works i
the awareness of the reader.
tion of typeset material, wit
of distractions and anomali
producing clarity and transp

Typeface choice is the prima
text typography,Äìprose ficti
editorial, educational, religi
spiritual and commercial wr
differing characteristics and
of appropriate typefaces and
toric material established te
frequently chosen according
historical genre acquired by
accretion, with considerable overlap between historical periods.

Contemporary books are more likely to be set with state-of-the-art seriffed "text romans" or "book romans" with design values echoing present-day design arts, which are closely based on traditional models such as those of Nicolas Jenson, Francesco Griffo (a punch cutter who created the model for Aldine typefaces), and Claude Garamond. With their more specialized requirements, newspapers and magazines rely on compact, tightly fitted seriffed text fonts specially designed for the

**Figure 1**: Inline styles (run-in paragraph heads) being created in a clip from the video *Typeset a page in under 10 minutes in InDesign.*

The video (Figure 1, `youtube.com/watch?v=VVIQE6kht8c`) is speeded up for brevity, and has a lot of clicking on and off of menus, selecting options and values. It starts as a single-column text document on Letter paper, and they import the material from an external plain text file all about typography. The text is set in Minion Pro and is edited to elide multiple spaces, and then cast into two columns...and then three, then the size is changed from 10pt to 12pt to 8¼pt to see the effect. Styling follows, first with paragraph heads, then with section heads, and finally the title. There is a lot of adjustment to the appearance of the paragraphs: changing the set and the spacing; but the end result is very good, and makes an excellent one-off example document to sell people on the idea that InDesign makes it easy to lay out one-off documents.

A few stop–start viewings made it clear that most of the formatting is fairly standard, and can easily be done in LaTeX using standard packages. I'm not sure about the practice of narrowing the set within a multi-column page on a column-by-column basis (the objective seems to be to regularize some parts of the ragged-right setting), but without any narration, the objectives are sometimes unclear.

To test this, I made a small proof-of-concept using the Wikipedia text on *Typography*, but there is significant editing work needed to get it into line with the text used in the video, and a need for abilities in creating a video from screen captures which are outside my skill-set. If there is a LaTeX user out there who is also fluent in InDesign, I'd be interested to see someone dissect the video with a view to making one for LaTeX, as I think it would be a good example for users to see a real-life document being brought into a popular format.

### Footnotes as never before or since — a text challenge

A historian on Twitter tweeted (at `twitter.com/garius/status/1570771789827166208`):

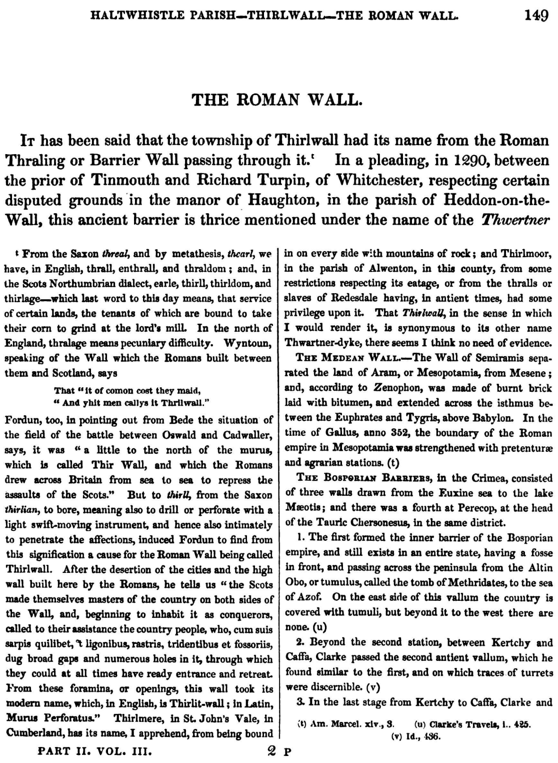
> It's Friday. Have some history.
>
> So you know Hadrian's Wall? Well for over 1000 years everyone thought it was built by someone else.
>
> Until, in 1840, John Hodgson, an unknown Northumbrian clergyman published the LONGEST footnote in history.

It ran to 173 pages. After the footnote, the main text simply carries on with the subject of the local history as if nothing had happened. If you can't read the thread, there's a spoiler in this footnote.[2]

[1] InDesign provides access to its own Document Object Model (DOM) so in theory you can use other scripting languages like AppleScript or VBScript.

[2] Hodgson was embarrassed by the fact that his *magnum opus* on Northumbrian history would likely be overshadowed

The book [1] is now online in Google Books at `books.google.co.uk/books?id=D1IGAAAAQAAJ`, where you can download a PDF; and there on p.149 Hodgson starts his explanation (Figure 2).



HALTWHISTLE PARISH—THIRLWALL—THE ROMAN WALL. 149

THE ROMAN WALL.

It has been said that the township of Thirlwall had its name from the Roman Thraling or Barrier Wall passing through it.[t] In a pleading, in 1290, between the prior of Tinmouth and Richard Turpin, of Whitchester, respecting certain disputed grounds in the manor of Haughton, in the parish of Heddon-on-the-Wall, this ancient barrier is thrice mentioned under the name of the *Thwertner*

[t] From the Saxon *threal*, and by metathesis, *thcarl*, we have, in English, thrall, enthrall, and thraldom; and, in the Scots Northumbrian dialect, earle, thirll, thirldom, and thirlage—which last word to this day means, that service of certain lands, the tenants of which are bound to take their corn to grind at the lord's mill. In the north of England, thralage means pecuniary difficulty. Wyntoun, speaking of the Wall which the Romans built between them and Scotland, says

> That "it of comon cost they maid,
> "And yhit men callys it Thrilwall."

Fordun, too, in pointing out from Bede the situation of the field of the battle between Oswald and Cadwaller, says, it was "a little to the north of the murus, which is called Thir Wall, and which the Romans drew across Britain from sea to sea to repress the assaults of the Scots." But to *thirll*, from the Saxon *thirlian*, to bore, meaning also to drill or perforate with a light swift-moving instrument, and hence also intimately to penetrate the affections, induced Fordun to find from this signification a cause for the Roman Wall being called Thirlwall. After the desertion of the cities and the high wall built here by the Romans, he tells us "the Scots made themselves masters of the country on both sides of the Wall, and, beginning to inhabit it as conquerors, called to their assistance the country people, who, cum suis sarpis quilibet, ⁊ ligonibus, rastris, tridentibus et fossoriis, dug broad gaps and numerous holes in it, through which they could at all times have ready entrance and retreat. From these foramina, or openings, this wall took its modern name, which, in English, is Thirlit-wall; in Latin, Murus Perforatus." Thirlmere, in St. John's Vale, in Cumberland, has its name, I apprehend, from being bound in on every side with mountains of rock; and Thirlmoor, in the parish of Alwenton, in this county, from some restrictions respecting its eatage, or from the thralls or slaves of Redesdale having, in antient times, had some privilege upon it. That *Thirlwall*, in the sense in which I would render it, is synonymous to its other name Thwartner-dyke, there seems I think no need of evidence.

The Medean Wall.—The Wall of Semiramis separated the land of Aram, or Mesopotamia, from Mesene; and, according to Zenophon, was made of burnt brick laid with bitumen, and extended across the isthmus between the Euphrates and Tygris, above Babylon. In the time of Gallus, anno 352, the boundary of the Roman empire in Mesopotamia was strengthened with pretenturæ and agrarian stations. (t)

The Bosporian Barriers, in the Crimea, consisted of three walls drawn from the Euxine sea to the lake Mæotis; and there was a fourth at Perecop, at the head of the Tauric Chersonesus, in the same district.

1. The first formed the inner barrier of the Bosporian empire, and still exists in an entire state, having a fosse in front, and passing across the peninsula from the Altin Obo, or tumulus, called the tomb of Methridates, to the sea of Azof. On the east side of this vallum the country is covered with tumuli, but beyond it to the west there are none. (u)

2. Beyond the second station, between Kertchy and Caffa, Clarke passed the second antient vallum, which he found similar to the first, and on which traces of turrets were discernible. (v)

3. In the last stage from Kertchy to Caffa, Clarke and

(t) Am. Marcel. xiv., 3. (u) Clarke's Travels, i.. 425.
(v) Id., 436.

PART II. VOL. III. 2 P

**Figure 2**: The start of the longest footnote.

As with the InDesign video, I had to see what LaTeX would make of 173 pages of a single footnote, but *pdftotext* does not convert Google's PDFs to plaintext well, as multiple lines of text from one column get interleaved with similar-sized blocks of text from the other, despite the vertical rule. Unfortunately Apache *PDFbox* fared no better, interleaving almost every single pair of lines. This appears to be a problem with the way in which the Google scan and OCR have been synchronised, presumably done for the purposes of searching, not reprocessing.

Looking through the pages, there are footnotes within the footnote; run-in subheadings; lists; bibliographic references between paragraphs; tabular settings; embedded graphics, including typographic reconstructions of inscriptions across both columns; and genealogical tree-charts — in fact almost the entire panoply of typographic requirements needed for a whole book.

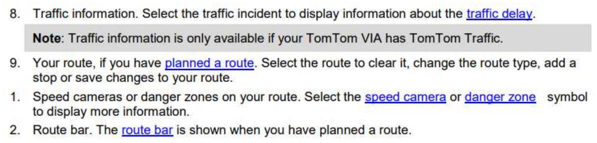
by including his discovery that the Roman wall everyone until then thought was built by Septimius Severus was actually built by Hadrian, so he put it in a footnote.

By comparison, the page layout itself would be unproblematic, except that the dblfnote package appears to reset to single-column after a page-break, and would in any case need modifying to allow reversion to single-column mode in mid-page for an illustration, like multicol does.

However, the simple answer to my original question is that LaTeX has no problem whatever in maintaining scope for 173 pages of footnote. However, a lot of work would be needed for a full reconstruction to test if all the formatting can be done, if someone [else] would like to take it up.

## Afterthought: List spacing

Just browsing the documentation for an old TomTom VIA 52 satnav and discovered that the main screen diagram has eleven callouts (icons labelled 1–11) but the list explaining them runs from one to nine, and then restarts at one (Figure 3).



8. Traffic information. Select the traffic incident to display information about the traffic delay.

**Note**: Traffic information is only available if your TomTom VIA has TomTom Traffic.

9. Your route, if you have planned a route. Select the route to clear it, change the route type, add a stop or save changes to your route.
1. Speed cameras or danger zones on your route. Select the speed camera or danger zone symbol to display more information.
2. Route bar. The route bar is shown when you have planned a route.

**Figure 3**: Callout list misnumbered

It looks very much as if the width of the indent provided for the list geometry was set to allow just a single digit … but if that was so, truncation on the left-hand side would have given zero (from the 10) and one (from the 11); truncation on the right-hand side would give two ones. So we must conclude that at 10, the list was actually reset to start at one. Just a little thing to be aware of when you design list indents. And, of course, use LaTeX, not Word, InDesign, or anything else.

## References

[1] J. Hodgson. *History of Northumberland.* [Published] for the Author, Newcastle, Jun 1840.

[2] Type-Ed. Typeset a page in under 10 minutes in InDesign, May 2016. `youtube.com/watch?v=VVIQE6kht8c`

◇ Peter Flynn
Textual Therapy Division,
Silmaril Consultants
Cork, Ireland
Phone: +353 86 824 5333
`peter (at) silmaril dot ie`
`blogs.silmaril.ie/peter`

# An artist's journey on a TUGboat

Tine Wilde

## Abstract

How does a coloured bird end up on a TUGboat? This is the story of an artist who studied philosophy and combined her skills in a PhD at the University of Amsterdam (NL). In order to write her dissertation, she had to learn the LaTeX typesetting programme. Many years later, she still makes art and still writes down her thoughts in LaTeX, with the Memoir class and XeLaTeX as first choice. Always trying to stretch the limits of the programme to her convenience.

This essay is not a technical article written by a developer conjuring up ingenious innovations to the LaTeX programme. By contrast, it discusses the relationship between LaTeX, art (photography) and the concept of 'measurability' from the perspective of a philosophising artist.

## 1 Introduction

After working as a visual artist for some time, I decided at some point to take my work to the next level with a study in philosophy. I only planned for a Master's degree, but things would turn out differently. Thanks to my supervisor Martin Stokhof, I got the opportunity to extend my research in a PhD project which would enable me to combine my artwork with philosophical insights through the ideas of Ludwig Wittgenstein. Since the study of Wittgenstein was part of analytic philosophy, and analytic philosophy was part of the Institute for Logic, Language, and Computation (`www.illc.uva.nl`), I was housed together with logicians and computer specialists. A preliminary condition was that every doctoral candidate would write their dissertation in the LaTeX typesetting programme. Thus, so did I.

While most of the PhD students used the LaTeX programme to typeset mathematical or logical formulas, i.e., the scientific writing stuff, I was allowed to employ it freely and without any limitations. Excellent, yet I had no idea of any typesetting programme to begin with. I had written my Master's thesis in Word. A programme I disliked, mainly because of all sorts of instabilities. An alternative was highly appreciated, but ... what was LaTeX?? I started out by buying *The LaTeX Companion* [6] in which numerous issues on fonts, tables, colours, and abbreviations were spelled out. The book is still on my desk after all these years and now reads more like a bible, although much can also be found online on the internet these days. Subsequently, I installed TeXShop and BibTeX on my Mac so that it began to look like a complete typesetting programme.

What helped and still helps me most is the way in which the programme forces one to structure the writing, and, by consequence, the thinking. Although it took some effort to learn how everything was set up and could be utilised, I loved the outcome. The results looked great, and, no instabilities in contrast to the Microsoft Word programme. Another quality I had never encountered elsewhere was the fact that I could add text without actually showing it in the final PDF file. A wonderful feature for storing remarks that were intended for my eyes only. Furthermore, the BibTeX bibliography turned out to be a marvellous tool. But what struck me most was the fact that all this was open source: made to be used — not to be bought and sold. A concept that needs to be cherished, in my view. There was just one problem left: the artistic world and more generally the world outside academia were by and large ignorant or reluctant to adopt the LaTeX programme and its benefits.

Printers specialised in art usually work with the Word programme and transfer the text, after editing and proofreading, into InDesign. For them, a LaTeX programme is beyond their scope. To some, it is a serious threat, while others declared it something weird from Mars. And me somewhere in between, being tossed around on a TUGboat. The two worlds clashed on this issue. I was keen on working with the LaTeX programme, *and* I wanted my dissertation to be printed by a renowned printer specialised in art. In the end, the printer and I agreed to use the LaTeX files instead of InDesign, but not until after some fierce discussion and deliberation [7].

I kept using LaTeX after having finished my PhD, and extended my LaTeX skills with the Memoir class and XeLaTeX. Playing with the system and producing all kinds of templates to make life easy since my projects, merging philosophical insights with artistic outcome, are complex and more often than not take several years to complete. In order to keep track, I need text in which I can add many references and cross-references, as well as private considerations. In case I have to apply for funding at multiple organisations, these elaborated texts help me get my insights across concisely.

Yet, there is another way in which the typesetting programme is helpful. It stimulates me to think about the ideas concerning creative processes, the nature of 'reality', and the concept of 'measurability'. In the next section, I will give you an illustration of how the LaTeX programme intrinsically has found its way into my artistic practice.

## 2 Measurability

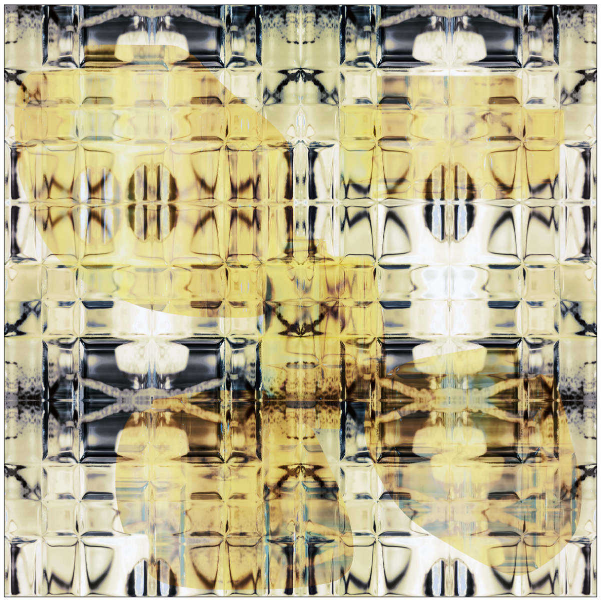
Within my research into the nature of ‘reality’, what fascinates me as a visual artist as well as a philosopher is the notion of measurability. The boundaries between fixed and fluid; between sharp and vague; between coloured and non-coloured; between love and hate. In short, I would ask myself: when does one state of affairs turn into the other? When is something still measurable? And: is it important for something to be measurable at all? Trying to shed light on these questions, I merge art (photography) with philosophy and theoretical considerations from physics. Allow me to give you an example by highlighting a project I have been working on since 2020.

### Project Zero Point

ZERO POINT #S2021-03k. Picturework, dimensions 71x71cm.

One single piece of glassware was taken as a starting point for transformational processes, in which photographic images were changed into thirty-eight multi-layered compositions distributed over four series. Pictorial spaces that are variable and subject to change in an experimental, unconscious method of choice, inspiration, and demolition. The results of split, remodel, and repeat invite you to explore and contemplate the notions of space and time as a dimension in which meaning remains something definitely unfinished, but in which measurability is crucial. After all, when there is nothing to hold on to, you have to choose a point of departure from which you can (re)organise your life.

The most significant literature and validation for project Zero Point is the work of David Bohm (1917–1992) with his research into the underlying meanings concerning quantum theory. As a theoretical physicist, he was one of the founders of quantum physics who worked under Oppenheimer on the Manhattan Project and collaborated at Princeton with Einstein. In an attempt to combine relativity and quantum theory, he discusses in *Wholeness and the Implicate Order* [3] in what ways reality lies beyond appearances. Speculating that the proton takes the form of a wave collapsing inward in and expanding outward from all space, rather than being a solid, continuous particle in spacetime.


ZERO POINT #S2021-03l. Picturework, dimensions 71x71cm.

There is a hidden regime of reality, Bohm says, that is and always will be inaccessible to us. From this deeper order he calls the ‘implicate order’ the ordinary notions appear in the ‘explicate order’, i.e., the world as we know and experience it with our senses. The model for this proposed implicate order is not based on ‘things’ considered as objects, or phenomena in a Cartesian space-time order, but rather a lens-like flowing of simultaneously enfolding and unfolding dimensions between an implicate and an explicate order.

Simply put, the implicate order is the ‘ground’ within which the entire universe is enfolded at each ‘point’ in spacetime, manifesting itself in an explicate unfolding order of a world we can see, hear, smell, touch and feel. It is an explicate order and for us actually present as a direct surface order. Underneath, there is a deeper, for humans only indirectly knowable or inaccessible order. A multidimensional sea of energy as an implicate order from which particles and spacetime can arise. This underlying reality and the explicate order as we experience it daily are intrinsically interwoven, amounting to one non-local, non-analysable breath breathing system, enfolding and unfolding at every moment into a complex, never fully knowable totality.

Bohm's concept of 'hidden variables' [1, 2] operates at the boundary of spacetime: "We come to a certain length at which the measurement of space and time becomes totally indefinable. Beyond this, the whole notion of space and time as we know it would fade out into something that is at present unspecifiable. So, it would be reasonable to suppose, at least provisionally, that this is the shortest wavelength that should be considered as contributing to the 'zero point' energy of space. When this length is estimated, it turns out to be about $10^{-33}$cm. This is much shorter than anything thus far probed in physical experiments (which have gone down to about $10^{-17}$cm or so). If one computes the amount of energy that would be in one cubic centimeter of space, with this shortest possible wavelength, it turns out to be very far beyond the total energy of all the matter in the known universe." [3, p. 190]. David Bohm called this calculation the 'zero-point' energy for a point of space. Here, he predicts, will be found a boundary separating an outer, explicate order.

In a 1987 paper [4], Bohm suggests that there may be multiple explicate orders as suborders of a single, infinitely connected implicate order. "A kind of universal process of constant creation and annihilation, determined through the super-quantum potential, so as to give rise to a world of form and structure in which all manifest features are only relatively constant, recurrent and stable aspects of this whole." [4, p. 43]. Further elaboration on this topic would exceed the scope of this article. A good introduction to Bohm's thoughts on the subject is now recorded on film and can be found at `www.infinitepotential.com`. On this website, you also will find additional illuminating interviews with Roger Penrose and Basil Hiley, among others.

David Bohm's search for a new notion of order amounts to a 'no final form' of insight. He urges us to view the world not as being constituted by basic objects or building blocks, but in terms of a universal flux of events and processes. Each relatively autonomous and stable structure should be understood as a product that has been formed in the whole flowing movement that will ultimately dissolve back in this moment. How it forms and maintains itself depends on the place and function in the whole.

## Photography and Philosophy

We, as human beings, are placed on a planet where we are made up of, but also are dependent on, our ability to recognise patterns on the basis of similar differences and different similarities. First perceptually: 'unguided' we recognise patterns in the blots on a brick wall or on wallpaper. Leonardo da Vinci already pointed at this faculty as a starting point for all art. Or in rituals in which a particular organisation of lines and forms produces a mesmerising effect, like a mandala or a kōlam. Second, as an attempt to make human knowledge measurable through 'information', that is, every amount of data, code, or text that can be preserved, sent, received or manipulated in any medium. Consider, for example, the mathematical numbers and proportions that can be grasped by the mind. Plato stated that these abstractions were of a higher order than the phenomena. According to him, the senses merely produce opinions, whereas the abstractions deliver certain, that is, 'true' and 'perfect' knowledge. In later times, this idea was refuted and certainty was exchanged for probability. We have now reached the point at which we are slowly coming to understand that true knowledge means insight. Insights that are neither true nor false, but continuously illuminate different aspects of specific regions or frameworks. In consequence, we will never reach crystal-clear conclusions.

The all-encompassing truth about the universe, then, is enclosed in the possibilities and constraints of the human powers of imagination. In the end, even a scientific system is but a free play with symbols according to (logical) arbitrarily given rules of the game — a free invention. "Thinking without the positing of categories and of concepts in general would be as impossible as breathing in a vacuum," as Einstein would put it [5, p. 674]. Our concepts are tools, of which we have to assume that they will behave differently in different domains. On the other hand, rule following constitutes a general framework in which it is possible to compare various (language) games, according to Wittgenstein [7]. Experience and knowledge are interconnected and undivided activities, and, as a result, continuously susceptible to change and adjustments.

Already at a very young age, I was fascinated by the patterns that could be detected on the medallion wallpaper of my aunt. At that time, we spent our summer holidays with a reformed family who were living on the south coast of the Netherlands. There was only beach and sea and family members — no newspaper nor any radio or television set. Bored stiff, I began to use the small camera my father had bought me as a possibility to escape the somewhat restricted atmosphere. Catching the various patterns on the wallpaper on camera — whether distorted or not — made me invent all sorts of landscapes, faces, ghostly suggestions and the stories that came with them. I did not think these were of the same quality as the stories told by Star Trek or The Jacksons — for me the ultimate TV series back then — but at least they provided a way out of boredom. It was only much later that I began to appreciate these lonely holidays and the ways in which my imagination had been triggered by camera and wallpaper.

It is no coincidence, then, that later in life my favourite medium to work with as an artist became photography. More specifically, the 'digital patterns' that make up for the images. It is the main reason why I call my photographic work 'pictureworks'. Not questioning the image as we perceive it in documentaries, reports, or events, but investigating *how* an image may appear to us. As a result, no single decision is conclusive, but understood as part of a series of clear and explicit 'quantum-decisions'. In this conception, the ideas of the photographer are intrinsically tied up with the hard-wired, pre-programmed 'information' inside the camera, amounting in a joined venture to the final results.

Consequently, the pictureworks are not representations, but rather the energy of an unseen and unknown world in which the camera acts as a concentrated point of consciousness, trying to locate the unknown in a reciprocal poetic resonance between the explicate structures of the ordinary world and the implicate processes of the human soul. In the dynamics between the explicate and the implicate, just like between the seeing and the thinking, pictureworks, whether they are presented as a single work of art, a choice sequence or an installation, are not a point of view, but a field of perception and cognition that tries to connect us with the deeper levels of life: the big unanswerable questions, the mysteries. From this, then, photography is understood as a reflective and analytic 'philosophical' medium. The pictureworks have nothing to do with reports, stories, documentaries, registered events, and the like. Rather, they originate in images, taken from everyday reality and used as raw material, to be transformed into works of art that seek to touch upon the viewer's infinite number of subtle feelings.

More thoughts on the subject are in a paper I wrote in 2021 for *Pari Perspectives* [8]. The paper as well as a short video about the making of project Zero Point can be accessed from my website.

## 3 Open Source–Open Mind

Within the undivided wholeness of flowing movement, we make a move and move around, constantly creating some order that structures our everyday life. More or less in the same way as the LaTeX typesetting programme structures my thoughts into words and sentences and orders them into a comprehensible whole, so as to be able to communicate with others. Therefore, a huge THANK YOU to all the developers and people who in one way or another contribute to the open-source environment of the LaTeX typesetting programme, offering their spare time to adjust, improve, and alter the typesetting system. In the end, LaTeX is not about a system, or any language for that matter, but about creative people, trying to come up with solutions which will make the world a somewhat more open and more interesting place to be.

### Pictureworks

Zero Point #S2021-03k and Zero Point #S2021-03l were printed by courtesy of the artist ©2022 Tine Wilde c/o Pictoright, Amsterdam.

### References

[1] D. Bohm. A Suggested Interpretation of the Quantum Theory in Terms of "Hidden" variables. I. *Physical Review* 85(2):166–179, Jan. 1952.

[2] D. Bohm. A Suggested Interpretation of the Quantum Theory in Terms of "Hidden" variables. II. *Physical Review* 85(2):180–193, Jan. 1952.

[3] D. Bohm. *Wholeness and the Implicate Order.* Routledge & Kegan Paul Ltd., 1980.

[4] D. Bohm. Hidden variables and the implicate order. In *Quantum Implications. Essays in honour of David Bohm*, B. Hiley, F. D. Peat, eds., ch. 2, pp. 33–45. Routledge & Kegan Paul Ltd., 1987.

[5] A. Einstein. Reply to Criticisms. In *Albert Einstein: Philosopher–Scientist*, P. Schlipp, ed., The Library of Living Philosophers, pp. 665–688. La Salle, Ill.: Open Court, 1949.

[6] F. Mittelbach, M. Goossens. *The LaTeX Companion.* Addison-Wesley, 2nd ed., 2004.

[7] T. Wilde. *Remodel[l]ing Reality. Wittgenstein's übersichtliche Darstellung & the phenomenon of Installation in visual art.* Wilde Oceans, 2008.

[8] T. Wilde. An Enquiry into the Nature of our Relationship with Reality. *Pari Perspectives* Issue 10(Consciousness):122–128, Dec. 2021.

◇ Tine Wilde
`message (at) tinewilde` dot `com`
`https://www.tinewilde.com`
ORCID 0009-0008-7390-447X

# The DuckBoat — Beginners' Pond: No more table nightmares with `tabularray`!

Herr Professor Paulinho van Duck

## Abstract

In this installment, Prof. van Duck will introduce you to `tabularray`, a package for typesetting tabulars and arrays with LaTeX3.

## 1 Reputation record!

Hi, (La)TeX friends!

A sensational event happened on August 3rd, 2022. Prof. Enrico Gregorio, a.k.a. egreg, reached one million reputation points on TeX.SE, quack! He is the first TeX user to smash this record.

You can understand the exceptionality of this fact since on the leading site, Stack Overflow, fewer than ten people have passed that threshold.

I am pleased to have had the occasion to congratulate him personally. We met last summer at the seaside and had a pizza (without pineapple) to celebrate.

In the same period, the funny TikZpingus package [3] was created. It is a sort of TikZlings spin-off and allows you to draw nice penguins with a very wide set of features.

The picture above is an example of its use.

Now let us move to our current topic. Are you struggling trying to set the height of your table rows? Do you think aligning some cells at the top and some others at the bottom is a mission impossible? Are you bored with typing `\hline` at every row end? Are you going crazy vertically centering a text in a `\multirow`? Would you like to use colors and `booktabs` together?

Do you dream of a package that makes all these amenities with simple options? Sometimes dreams come true, quack!

I am pleased to introduce you to `tabularray`, a recent package for typesetting tabulars and similar environments with many handy options.

Of course, I will not explain all the features of the package, they are a *large* number! Please refer to the package documentation [2] for further details.

Last but not least, let me thank Jianrui Lyu, the brilliant author of the package, for his very accurate review of this article. He is also a user of TopAnswers TeX (`topanswers.xyz/tex`). If you ask a question there, you may get an answer directly from him.

I am also grateful to samcarter and egreg for their suggestions. samcarter also provided the example in Box 13.

Of course, any errors that remain are my sole responsibility.

## 2 Quack Guide n. 8: The `tabularray` package

As usual, the first thing to do is to load this awesome package in your preamble with
`\usepackage{tabularray}`

Its main environment is `tblr`; it works in both text and math mode. This is already a nice feature, is it not? You can also have a text table in a math environment by setting `mode=text` (more in Section 2.8).

The environment `tblr` has a mandatory argument, where you can specify all the options you like. For example, the standard `l` (left), `c` (center), and `r` (right) can be used for the horizontal alignment; there is also `j` for justified text.

It also has an optional argument, where, for instance, you can choose the baseline of the table. It will not be treated here for reasons of brevity; please refer to the manual [2].

In some of the examples shown in the article, vertical rules are used, but only to better show some `tabularray` features; remember: ***avoid vertical rules in professional tables***, they are generally useless and inelegant, quack!

### 2.1 Row and column separation (and some other options)

Box 1 shows the difference in default row separation between an ordinary `tabular`/`array` (on the left in the box) and a `tblr` (on the right).

The tables created by `tblr` look better because they have some extra space above and below the rows. You no longer have to fight against `\arraystretch` or `\extrarowheight`, quack!

The size of the vertical space above/below the row (or both) is fully customizable, respectively setting `abovesep`, `belowsep` or `rowsep` as options in `rows=`⟨*styles*⟩.

There are also the corresponding options for `columns` (`leftsep`, `rightsep`, `colsep`, with the obvious meanings).

You can even set the spacing (or any other option) for one or a group of rows/columns only.

With `row{`⟨*number*⟩`}={`⟨*styles*⟩`}` you can set options valid only for the rows indicated by ⟨*number*⟩.

**Box 1 – tabular and array vs. tblr**

```
\begin{tabular}{lcr}
  \hline
  p & v & d \\\hline
  p & dl & q \\\hline
\end{tabular} \hspace{1em} vs.\ \hspace{1em}
\begin{tblr}{lcr}
  \hline
  p & v & d \\\hline
  p & dl & q \\\hline
\end{tblr}\par\vspace{1ex}
$\begin{array}{cc}
  \dfrac{1}{2} & \dfrac{3}{4}  \\
  \dfrac{5}{7} & -\dfrac{9}{10} \\
\end{array}$ \hspace{1em} vs.\ \hspace{1em}
$\begin{tblr}{cc}
  \dfrac{1}{2} & \dfrac{3}{4}  \\
  \dfrac{5}{7} & -\dfrac{9}{10} \\
\end{tblr}$
```

| p | v | d |
|---|---|---|
| p | dl | q |

vs.

| p | v | d |
|---|---|---|
| p | dl | q |

$\begin{array}{cc} \dfrac{1}{2} & \dfrac{3}{4} \\ \dfrac{5}{7} & -\dfrac{9}{10} \end{array}$ vs. $\begin{array}{cc} \dfrac{1}{2} & \dfrac{3}{4} \\[1ex] \dfrac{5}{7} & -\dfrac{9}{10} \end{array}$

**Box 2 – Column and row options**

```
\begin{tblr}{colspec={llccc},
  rows={m, rowsep=2pt},
  columns={colsep=3pt},
  row{1,Z}={font=\bfseries, abovesep=3pt,
    belowsep=1pt},
  column{Z}={rightsep=0pt, fg=red},
  column{1}={leftsep=0pt},
  column{3-Z}={yellow!10}}
\hline
{First\\ name}&{Last\\ name}& A & B &
    Average\\\hline
Paulinho & van Duck & 10 & 20 & 15\\
Paulette & de la Quack & 30& 40 & 35
    \\\hline
Total && 40 & 60 & 50\\\hline
\end{tblr}
```

| **First name** | **Last name** | **A** | **B** | **Average** |
|---|---|---|---|---|
| Paulinho | van Duck | 10 | 20 | 15 |
| Paulette | de la Quack | 30 | 40 | 35 |
| **Total** | | **40** | **60** | **50** |

This could be a single number, a range ⟨*n–m*⟩, a list ⟨*n,m,p,q*⟩, or `even` or `odd` if you need some customizations for even or odd rows.

`odd[`⟨*n–m*⟩`]` and `even[`⟨*n–m*⟩`]` also accept an optional argument which specifies from/to which row you would like the ⟨*styles*⟩ to apply. If the end row is the last of the table, it can be omitted.

There is also the key for specific columns:
`column{`⟨*number*⟩`}={`⟨*styles*⟩`}`

You can also mix `rows`/`columns` for general settings and `row`/`column` for specific ones.

Please note that `tabularray` generally ignores spaces around and within its arguments. For instance,
`column{3-Z}={yellow!10}`
is the same as
`column {  3-Z } = { yellow!10 };`
only the range cannot be separated by spaces.

Another wonderful feature is the possibility to use `U`, `V`, `W`, `X`, `Y`, and `Z` as row/column numbers, to indicate the last six rows/columns, in the order. It is very convenient when you are defining a different style for the last rows/columns of your table but you do not know or do not want to count the number of rows/columns. Please be aware that the values `U`, `V`, and `W` were added in a very recent version of the package. Remember to update your TEX distribution to enjoy them, quack!

Box 2 shows an example. Please note that when other parameters are present, the column alignment must be specified using `colspec`. The first and the last row font is set to bold (with `font=`⟨*font commands*⟩). Columns from the third to the last have a background color, which can simply be set with ⟨*color*⟩, without specifying the key `bg`. For the foreground color, on the contrary, the key `fg=`⟨*color*⟩ is mandatory. Please remember to also load `xcolor` package, and use your imagination if you are reading the black-and-white version of this article.

You can also note the multiline cells simply created with `{...\\...}`. Goodbye, `\makecell`!

For nearly all the parameters you can set in the mandatory argument there are corresponding commands you can use inside the table contents. For example, `\SetRow` is like `row`. For the sake of brevity, with few exceptions, I will not show these commands here, please refer to the manual [2] for them.

## 2.2 No more pain with vertical alignment

The package `tabularray` allows very easy alignment setting.

**Box 3 – Vertical alignment with reference to baseline**

```
\begin{tblr}{colspec={Q[l,t]Q[c,m]Q[r,b]},
    hlines}
{Baseline is\\ the top line\\ (left
    aligned)} &
{Baseline is\\ at the middle\\ (centered)} &
{Baseline is\\ the bottom line\\ (right
    aligned)}\\
\end{tblr}
```

| | | |
|---|---|---|
| Baseline is<br>the top line<br>(left aligned) | Baseline is<br>at the middle<br>(centered) | Baseline is<br>the bottom line<br>(right aligned) |

**Box 4 – Vertical alignment for non-typographers**

```
\begin{tblr}{colspec={Q[l,h]Q[c,m]Q[r,f]},
    hlines}
{At the head\\ (left aligned)} &
{At the\\ middle\\ (centered)} &
{At the foot\\ (right aligned)}\\
\end{tblr}
```

| | | |
|---|---|---|
| At the head<br>(left aligned) | At the<br>middle<br>(centered) | At the foot<br>(right aligned) |

Using its “Quack” column type `Q[`⟨*styles*⟩`]`, you can simultaneously indicate the vertical and horizontal alignments. No more “complex” options like `>{\centering\arraybackslash}` needed!

The vertical alignment can be set with respect to the baseline as usual, using `t` (the baseline is the top line), `m` (the baseline is at the middle), or `b` (the baseline is the bottom line).

An example is in Box 3 (the `hlines` option will be explained in Section 2.4).

Since the “baseline” concept is not easily caught by newbies, `tabularray` also allows the `h` (head) and `f` (foot) alignment, see Box 4.

Even more awesome is the possibility to combine the alignment as you like, as in Box 5. Please note that if you use the pipes `|` in `rowspec` they are horizontal lines.

## 2.3 Customizable cell dimensions

Within the styles of your column, you can set the column width; the option is `wd=`⟨*dimension*⟩, but `wd=` can often be omitted.

**Box 5 – Combined horizontal/vertical alignment**

```
\begin{tblr}{colspec={Q[l]Q[c]Q[r]},
  rowspec={|Q[t]|Q[m]|Q[b]|}}
{Top\\ left} & Top centered & Top right \\
Middle left & {Middle\\ centered} & Middle
    right \\
Bottom left & Bottom centered & {Bottom\\
    right} \\
\end{tblr}
```

| | | |
|---|---|---|
| Top<br>left | Top centered | Top right |
| Middle left | Middle<br>centered | Middle right |
| Bottom left | Bottom centered | Bottom<br>right |

The same is true for the cell height, with the analogous key `ht=`⟨*dimension*⟩.

These two parameters together allow you to create cells of the exact dimension you need, both horizontal and *vertical*. No more hacking with struts!

The sudoku scheme in Box 6 is an example of perfectly square cells. The `stretch=0` option is used to have the numbers perfectly centered.

The solution of the puzzle is left to the reader. I also have to mention a `sudoku` package [1] already exists, quack!

## 2.4 Lines as you desire

Box 6 is also an example of how you could easily customize table rules (a.k.a. lines).

With `hlines`/`vlines` you can draw with a single option all the horizontal/vertical rules. There is no more need to put `\hline` at the end of every line or a pipe `|` between the column types, although they are accepted as well.

The advantage of `tabularray` is that you can customize any given rule, or even part of one. With:
`hline{`⟨*hnumber*⟩`}={`⟨*vnumber*⟩`}{`⟨*styles*⟩`}`
you can specify for which horizontal rules ⟨*hnumber*⟩, and from/to which vertical rules ⟨*vnumber*⟩, your options apply. The ⟨*hnumber*⟩/⟨*vnumber*⟩ could be also a range ⟨*n*–*m*⟩ or a list ⟨*n*,*m*,*p*,*q*⟩. Please pay attention that, in this case, they are the *rule* numbers, not the row/column numbers as in `row` or `column`.

The analogous option for vertical rules is:
`vline{`⟨*vnumber*⟩`}={`⟨*hnumber*⟩`}{`⟨*styles*⟩`}`

There are plenty of options for setting the rule appearance: width (`wd=`⟨*dimension*⟩), shape (`solid`,

**Box 6 – A sudoku puzzle**

```
\begin{tblr}{columns={.5cm, colsep=0pt},
  rows={.5cm, rowsep=0pt},
  cells={c,m},hlines,vlines,stretch=0,
  hline{1,4,7,Z}={wd=1.2pt},
  vline{1,4,7,Z}={wd=1.2pt}}
1& &6& & &5&4& & \\
 & &9&1& &8& &5& \\
7&5& &9& & &1& &3\\
 & & &5& & &3& &9\\
2& & & &4& & & &1\\
8& &4& & &9& & & \\
5& &7& & &3& &2&6\\
 &6& &8& &1&5& & \\
 & &3&2& & &9& &7\\
\end{tblr}
```

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | | 6 | | | 5 | 4 | | |
| | | 9 | 1 | | 8 | | 5 | |
| 7 | 5 | | 9 | | | 1 | | 3 |
| | | | 5 | | | 3 | | 9 |
| 2 | | | | 4 | | | | 1 |
| 8 | | 4 | | | 9 | | | |
| 5 | | 7 | | | 3 | | 2 | 6 |
| | 6 | | 8 | | 1 | 5 | | |
| | | 3 | 2 | | | 9 | | 7 |

**Box 7 – Multirow comparison**

```
\begin{tabular}{|p{2.1cm}|p{2.1cm}|}
\hline
\multirow{2}{*}{Multirow cell}& With
    \texttt{tabular}\\\cline{2-2}
& A cell with two text lines \\\hline
\end{tabular}\par\vspace{1ex}
\begin{tblr}{columns={2.1cm}, hlines,
  vlines, cell{1}{1}={r=2}{l}}
Multirow cell & With \texttt{tblr}\\
& A cell with two text lines \\
\end{tblr}
```

| | |
|---|---|
| Multirow cell | With `tabular` |
| | A cell with two text lines |

| | |
|---|---|
| Multirow cell | With `tblr` |
| | A cell with two text lines |

`dashed`, `dotted`), color (`fg=`⟨*color*⟩). In Box 6, I use this to make the first and then every third rule (both horizontal and vertical) a little thicker.

But, in my opinion, the most awesome feature is the possibility to use `U`, `V`, `W`, `X`, `Y`, and `Z` to denote the last six lines, respectively (as was seen above for rows and columns, but now for rules). This allows you to completely customize your table within the mandatory parameter of the `tblr` environment without worrying about adding new rows/columns. It is very useful if you would like to create your own tabularray environment, see Section 2.8.

## 2.5 Multirow (and multicolumn) never so easy

In ordinary tabular environments, in the presence of multirow cells, the vertical alignment is often a pain in the . . . quack! One of the big strengths of `tabularray` is how it manages this alignment.

In traditional environments, when some cells have more than one line of text, it is often necessary to manually adjust the multirow text position. `tabularray` does it automatically: see Box 7.

Please also note the horizontal lines are correctly drawn without specifying the columns where they should appear. The `multirow` package and `\cline` command are no longer needed!

The option to set a multirow/multicolumn is

`cell{`⟨*i*⟩`}{`⟨*j*⟩`}={`⟨*span*⟩`}{`⟨*styles*⟩`}`

where ⟨*i*⟩ and ⟨*j*⟩ are the row and column numbers (top left position of the multirow/multicolumn cell); ⟨*span*⟩ indicates how many rows (`r=`⟨*number*⟩) and/or how many columns (`c=`⟨*number*⟩) the cell should span; and ⟨*styles*⟩ gives the options of the multicell.

If you prefer, you can use an analogous command at the cell position: `\SetCell[`⟨*span*⟩`]{`⟨*styles*⟩`}`

When you merge cells horizontally, please note that you still need to give all necessary `&` characters for the empty cells. This is different behavior compared to the ordinary `\multicolumn`. If you omit any, the content of some of your cells might be missing. It can be difficult to detect, as it does not cause any error, only wrong output.

Another outstanding feature is the possibility of indicating the span algorithm.

For horizontal spanning, you can choose

`hspan=`⟨*algorithm*⟩

where ⟨*algorithm*⟩ can be `default` (the last column is enlarged to reach the width of the multicolumn cell), `even` (all the columns are equally enlarged to reach the width of the multicolumn cell), or `minimal`

**Box 8 – Horizontal spanning**

```
\begin{tblr}{hlines, vlines,
  cell{2}{1}={c=3}{l}}
First & Second & Third \\
Multicolumn cell with default span \\
\end{tblr}
\par\vspace{1ex}
\begin{tblr}{hlines, vlines, hspan=even,
  cell{2}{1}={c=3}{l}}
First & Second & Third \\
Multicolumn cell with even span\\
\end{tblr}\par\vspace{1ex}
\begin{tblr}{hlines, vlines, hspan=minimal,
  cell{2}{1}={c=3}{l}}
First & Second & Third \\
Multicolumn cell with minimal span\\
\end{tblr}
```

| First | Second | Third |
|---|---|---|
| Multicolumn cell with default span | | |

| First | Second | Third |
|---|---|---|
| Multicolumn cell with even span | | |

| First | Second | Third |
|---|---|---|
| Multicolumn cell with minimal span | | |

(the multicolumn cell is split on more lines to fit the width of the columns). An example is in Box 8.

For vertical spanning, there is
`vspan=`⟨*algorithm*⟩
where ⟨*algorithm*⟩ could be `default` (the last row is stretched to reach the height of the multirow cell), or `even` (all the columns are equally enlarged to reach the length of the multicolumn cell). See Box 9.

## 2.6 Additional libraries

Even though `tabularray` has such a rich variety of features, you may well want to use it together with other traditional packages, such as `amsmath`, `booktabs`, or `siunitx`.

Since `tabularray` modifies some commands in those packages, to avoid potential conflict, you need to load them with `\UseTblrLibrary` command.

Box 10 shows an example with the `tabularray`'s libraries `booktabs` and `counter`. Unlike an ordinary table with `booktabs` rules, the background color touches them and the vertical rules are not discontinuous when they cross the horizontal ones.

The library `counter` allows you to modify some counters inside `tabularray` tables. In the example,

**Box 9 – Vertical spanning**

```
\begin{tblr}{hlines, vlines,
  cell{1}{1}={r=2}{c}}
{Multirow\\ cell\\ with\\ default span} &
    First\\
& Second \\
\end{tblr}\par\vspace{1ex}
\begin{tblr}{hlines, vlines, vspan=even,
  cell{1}{1}={r=2}{c}}
{Multirow\\ cell\\ with\\ even span} &
    First\\
& Second \\
\end{tblr}
```

| Multirow cell with default span | First |
|---|---|
| | Second |

| Multirow cell with even span | First |
|---|---|
| | Second |

I used the counter `\mycount` that is increased by 1 at every row of the table, headers excluded.

The package `tabularray` also provides some counters that can be used with no need to load the library. They are: `rowcount` (total number of rows), `rownum` (number of the current row), `colcount` (total number of columns), and `colnum` (number of the current column). The first two are used in the third column of the table in Box 10.

With the option `preto=`⟨*text*⟩ I avoided writing the commands for showing the counters at every row. I wrote them only in the mandatory argument, specifying with `cell` where they should be applied. Is it not great? Quack!

`preto` prepends text to the cell; to append text to the cell you can use `appto`. And if the content of your cell should be the argument of a command, you can use `cmd`.

## 2.7 `X` column type

Like `tabularx`, `tabularray` provides for
`X[`⟨*coeff*⟩,⟨*alignm*⟩,⟨*styles*⟩`]`
columns, but with extra gear. With the optional parameters, you can configure their alignment ⟨*alignm*⟩, their width ⟨*coeff*⟩, or other settings ⟨*styles*⟩.

The width is in a form of a multiplicative coefficient, as in the (outdated) package `tabu`. For example, `X[2,r]` is a right-aligned column with a

**Box 10 – booktabs and counter libraries**

```
...
\usepackage{tabularray}
\UseTblrLibrary{booktabs}
\UseTblrLibrary{counter}
\newcounter{mycount}
\newcommand{\myc}{%
  \stepcounter{mycount}\arabic{mycount}}
...
\begin{document}
\begin{booktabs}{vlines,
  colspec = {lcc}, hspan=even,
  cell{1}{1} = {r=2}{},
  cell{1}{2} = {c=2}{c},
  row{odd[3]} = {bg=cyan!10},
  cell{3-Z}{2}={preto={\arabic{rownum} of
    \arabic{rowcount}}},
  cell{3-Z}{3}={preto={\myc}}}
\toprule
Name & Counters \\
\cmidrule{2-3}
& {\texttt{rownum} of\\ \texttt{rowcount}}
    & \texttt{mycount} \\
\midrule
Paulinho van Duck & & \\
Paulette de la Quack & & \\
Paolino Quaqua & & \\
Pauline von Ente & & \\
Paulina Pato & & \\
\bottomrule
\end{booktabs}
\end{document}
```

| Name | Counters | |
|---|---|---|
| | `rownum` of `rowcount` | `mycount` |
| Paulinho van Duck | 3 of 7 | 1 |
| Paulette de la Quack | 4 of 7 | 2 |
| Paolino Quaqua | 5 of 7 | 3 |
| Pauline von Ente | 6 of 7 | 4 |
| Paulina Pato | 7 of 7 | 5 |

**Box 11 – `X` column type**

```
\begin{tblr}{width={.9\linewidth},
  colspec={X[2, font={\itshape}]
    X[2,c]X[-1]X[-1]XX[r]},
  vlines, hlines}
A & B & C & D & E & F\\
Q & u & a a a  & c & k & !!!\\
\end{tblr}
```

| *A* | B | C | D | E | F |
|---|---|---|---|---|---|
| *Q* | u | a a a | c | k | !!! |

doubled width, compared to the other `X[1]` columns of the table (`[1]` is the default and can be omitted).

`X` columns with negative coefficients are also possible. In this case, the columns have their "natural width", but are limited to the width of a corresponding `X` column with a positive coefficient. For instance, `X[-1]` is like an `l` column but if its width is greater than what an `X[1]` column would have had, then it is adapted to the `X[1]` column width. In Box 11, column C has the same width as E or F, whereas column D has its natural width.

The total width of the table is `\linewidth` by default. If you need a different width, use the option `width=`⟨*dimension*⟩.

## 2.8 New row/column types, new environments, and other tricks

As for ordinary environments, you can create your own column types with:
`\NewColumnType{`⟨*type*⟩`}[`⟨*n*⟩`][`⟨*dflt*⟩`]{`⟨*styles*⟩`}`
where ⟨*type*⟩ is one letter indicating the name of the new column type; ⟨*n*⟩ is the number of parameters, if any; ⟨*dflt*⟩ is the default parameter, if any; ⟨*styles*⟩ are the column options.

Since, with `tabularray`, you have row options, you can also create your own row types, with the analogous command `\NewRowType`.

In Box 12, column type `T` is defined as left aligned, text mode (`mode=text` allows to use a text column in a math environment); and `D` as centered display math (`dmath`) mode. The option `mode` also provides for `imath` and `math` for inline math mode.

As mentioned above, the possibility to have all the settings in the mandatory argument is particularly useful if you like to create your own table template, because styles are totally separated from the contents of your tables.

With `\NewTblrEnviron{`⟨*envname*⟩`}` you can define your own environment. Then, with
`\SetTblrInner[`⟨*envname*⟩`]{`⟨*styles*⟩`}`
you can set all the styles you like for the environment ⟨*envname*⟩. Here the environment name is optional; if you leave it out, the styles for all the `tblr` of your document will be set.

For example, `\SetTblrInner{rowsep=0pt}` sets the spacing as in the ordinary `tabular`, for all your `tblr` environments.

**Box 12 – New types and environment**

```
\NewColumnType{T}{Q[l, mode=text]}
\NewColumnType{D}{Q[c, mode=dmath]}
\NewTblrEnviron{mytab}
\SetTblrInner[mytab]{colspec={TD},
  row{1,Z}={font=\bfseries, mode=text},
  hline{1,2,Y,Z}={leftpos=-1, rightpos=-1,
    endpos},
        cells={m}}
\begin{mytab}{}
  Shape&Area\\
  Circle & \pi r^2 \\
  {Total rows of this table} &
    \arabic{rowcount}\\
\end{mytab}\par\vspace{1ex}
\begin{mytab}{rowsep=3pt}
  Shape&Area\\
  Square & a^2 \\
  Rectangle & w\cdot h\\
  {Total rows of this table}  &
    \arabic{rowcount}\\
\end{mytab}
```

| **Shape** | **Area** |
|---|---|
| Circle | $\pi r^2$ |
| **Total rows of this table** | **3** |

| **Shape** | **Area** |
|---|---|
| Square | $a^2$ |
| Rectangle | $w \cdot h$ |
| **Total rows of this table** | **4** |

There is also
`\SetTblrOuter[`⟨*envname*⟩`]{`⟨*styles*⟩`}`
to set the specifications of the optional argument of your environment, such as the baseline. Please refer to the package manual [2] for this.

Box 12 shows an example with the new environment `mytab`. Please note the use of the `Y` and `Z` in the `hline` and `row` options. This way you build a template that is always valid, regardless of the number of rows in the table.

The options `leftpos=-1` and `rightpos=-1` of `hline` mean that the lines are trimmed by `colsep` on the left and on the right; with `endpos` this adjustment is applied only to the leftmost/rightmost column.

The `tabularray` support for having all the definitions separated from the content turns out to be very helpful when you have to format tables for which you do not have easy access to the source code, e.g. because they are automatically generated by R or Markdown or another tool.

**Box 13 – A useful trick**

```
\documentclass{article}
\usepackage{tabularray}
\renewenvironment{tabular}[2][c]{
  \begin{tblr}[baseline=#1]{
    row{1-2}={font=\bfseries},
    colspec={#2}}
}{\end{tblr}}
\begin{document}
\begin{table}[htbp]
\centering
\input{tab.tex}
\end{table}
\end{document}
```

Imagine having the code of a table you cannot change, such as `tab.tex`, but you would like to have the first two rows in bold. With `tabularray` you could redefine the `tabular` environment adding any style you like. Box 13 shows how to do it.

## 3 Conclusions

I hope you liked the features supplied by this modern package, and if you have problems with your tables, remember:

***Try*** `tabularray`
***and all will be OK!***

## References

[1] P. Abraham. The sudoku package. Version 1.0.1. `ctan.org/pkg/sudoku`

[2] J. Lyu. Tabularray — Typeset Tabulars and Arrays with LaTeX3. Version 2023A. `ctan.org/pkg/tabularray`

[3] F. Sihler. The Ti*k*Zpingus package. Version 1.0. `ctan.org/pkg/tikzpingus`

◇ Herr Professor Paulinho van Duck
Quack University Campus
Sempione Park Pond
Milano, Italy
`paulinho dot vanduck (at) gmail dot com`

# Metadata in journal publishing

Joppe W. Bos, Kevin S. McCurley

## Abstract

We discuss how to use LaTeX classes and BibTeX styles to curate metadata throughout the life cycle of a published journal or conference article. Our focus is on streamlining and automating much of the publishing workflow.

## 1 Introduction

The original goal of TeX was to provide a system for typesetting, namely to control the layout of a document on paper. The later invention of LaTeX was focused on "letting the user concentrate on the structure of the text rather than on formatting commands" [8]. Users were encouraged to write their papers using high-level macros like `\section`, and leave the decisions like how much space to put before or after a section to the style that is used. As a result, an author does not have to worry so much about how the paper looks, but primarily about how the paper is logically structured.

This separation of concerns about appearance versus structure has proved to be very effective and most, if not all, scientific publishers now have their own LaTeX styles. These styles make it easy for an author to conform to a common look and feel in a journal, and can streamline the production steps for a journal if authors comply with the style. Moreover, it is usually easy for authors to convert from one style to another, because most of them adhere to standard macros like `\section`.

There is however at least one area in which the LaTeX community has been slow to adapt to the needs of modern publishing workflows, namely in the *curation of metadata about publications*. This is the main focus of this article.[1] We believe that a LaTeX style serves two roles; namely to provide a mechanism for describing structural information about the document, and a style for describing how to lay it out on the page.

## 2 Metadata in publishing workflows

When we refer to metadata, we include data objects such as title, subtitle, author names, e-mail addresses, ORCIDs, affiliations, funding agencies, bibliographic citations, journal identifier, page numbers, DOI, etc. Some of this metadata is supplied by the publisher at the time of publication, but much of it is supplied by the authors. Once the author submits their final version, this metadata is typically used to register for DOI, at which time the publisher needs to supply a considerable amount of metadata. Moreover, the web "landing page" for a paper typically has to be created from the metadata. Indexing agencies then step in, either by crawling the data or by receiving metadata feeds from the publisher. This metadata is crucial for ranking, indexing, and organization of scientific publishing.

### 2.1 Economics of publishing

Part of our motivation arises from our involvement in trying to launch a new open access journal for the professional non-profit society International Association for Cryptologic Research (IACR).[2] The society already runs two diamond open access journals, but experience from running these has shown that on average each published paper requires about an hour of human effort for production and metadata handling. Even then we find that errors sometimes slip through. Another study [4] estimated the amount of human labor for editing and production to be 7.5 person-hours for each published paper. We believe that most of this should and could be automated, and this can help to lower the cost of publishing. This is particularly important for open access publishing, which is heavily dependent on volunteer labor [1] as a way to control costs. It can also be used to improve profitability of commercial publishers.

In some systems, such as Open Journal Systems [12], the submission and curation of metadata is treated as a separate task from submission of the Word, LaTeX or PDF document. This imposes an extra burden on authors, and also renders the workflow vulnerable to inconsistencies with metadata in two places. In our experience, by the time an article has been revised and accepted, there are often changes in titles, abstracts, affiliations, email addresses, references, etc. Checking and correcting these inconsistencies ends up costing time of the human authors and editors.

For this reason, we believe that a LaTeX class should provide a convenient mechanism for authors to enter the metadata only once, in a standard way that encodes relationships between entities. From that point on, it should be possible to generate appropriate machine-parsable formats which can be used at every phase in the publishing pipeline.

## 3 Our approach at a high level

We automate the capture of metadata during the publishing workflow through the use of a LaTeX class

[1] An earlier and longer version of this article was published at arxiv.org/abs/2301.08277.

[2] See iacr.org.

doi.org/10.47397/tb/44-1/tb136bos-metadata

iacrcc.cls and a BibTeX style iacrcc.bst.[3] The function of these files is to both display the metadata in the output format, but to also extract the metadata during the compilation process, producing an easily parsable external format as a side product.

When authors supply their final versions, they do so by uploading their LaTeX source to a cloud server, which compiles their sources and extracts all metadata from their sources into a text file with a structured format (together with performing some sanity checks on the provided data). The submission process does not require authors to enter any additional metadata, because it is all encoded into the LaTeX source. The DOI suffix is assigned by the server and the DOI is compiled directly into the PDF at time of submission. A post-compilation step is used to parse the structured metadata and convert it into other formats, including JSON and XML. The DOI is registered with the DOI registration agency once the copyediting phase is complete. The extracted metadata is also used to produce various web pages for the journal site, RSS feeds, OAI-PMH feeds, and register with various indexing services.

The metadata output we require is necessarily *text*, and the lingua franca for encoding of text is UTF-8. With the exception of mathematical structures like inline equations in titles or abstracts, this text is devoid of TeX macros. This causes a few problems in the LaTeX world, which encourages authors to write in 7-bit ASCII text with user-defined macros.

Part of our problem arises from the fact that TeX takes the input format and produces a list of tokens. This sequence of tokens is convenient to produce a list of boxes containing glyphs for layout on pages, but extraction of the author's original text from that token list is problematic. For example, spaces are not space characters but are instead glue between boxes or terminators for macros.

In addition, a core functionality of LaTeX is user-defined macros, so an author might define `\pe` to represent the text string "Paul Erdős". We only discover this during the LaTeX expansion process when the macros are expanded into glyphs. Macro expansion is one of the most difficult topics in understanding how TeX works.

Our first implementation of metadata capture used the `\write` macro during the LaTeX compilation process to write an external file containing metadata. The intended function of the `\write` macro is to expand a list of tokens and write a parsable representation of these tokens into a file. The fact that `\write` performs expansion is very useful to us, because it expands user-defined macros. Unfortunately `\write` also causes a few problems when we use it to produce metadata. As an example, `\(` and `\)` cannot be used to delimit inline mathematics inside `\write`, whereas `$` works fine.

Another problem arises with `pdflatex`, because we have found examples like `\write{Đ and f\"ur}` where the output from `\write` contains mixed character encodings in a single line. This is apparently due to the fact that while `pdflatex` handles UTF-8 input, the output tries to use the single-byte Cork encoding for things like `ü`. For this reason we switched to using `\protected@write` instead of `\write`, following a suggestion from the LaTeX team.

One might argue that the author can correct the previous example by avoiding mixed encodings in their input, but this is merely one example of many ways that authors can produce legitimate LaTeX that is difficult to deal with. Our goal is to provide a system that supports whatever legitimate LaTeX the author supplies to us, and to provide them with clear instructions on how to prepare it without causing any interruptions (errors) or other inconvenience to the author's typesetting experience. From an author's point of view, the flexibility of LaTeX can be a blessing, but it's also often a curse for a journal.

### 3.1 Alternative approaches

We considered several ways to implement the metadata extraction instead of using `\write`. One alternative approach would be to use a LaTeX parser to extract the metadata directly from the LaTeX. The problem of parsing LaTeX is complicated by the need to expand macros, for which the LaTeX engines themselves are so far the only robust solution. Another approach that we considered involved using Lua within `lualatex`. Lua is much better suited to text processing than using LaTeX itself, but we had an initial goal to try and make things work with any LaTeX engine.

## 4 What metadata is required?

Some metadata fields in a journal article are obvious (title, author), but even the obvious fields have nuances in how they are encoded. Examples include:

- Title of the work. In some fields it is commonplace to use mathematics in titles, but TeX formatting in metadata records is often changed to another format like MathML. Titles may also encode face markup (e.g., bold face) or multiple

[3] The authoritative place to download these is publish.iacr.org/iacrcc.

character sets. Extremely long titles are sometimes broken up into a hierarchy, incorporating a subtitle or short versions for running titles.

- Authors of the work. One reason to ask for authors is to give proper attribution in citations, but author names are not unique so we should also use a unique identifier like ORCID.
- Authors may have different levels of contribution. In some cases this is signaled by having author names out of alphabetical order, but in other fields it is common to identify a *role* for author contributions. The CRediT taxonomy is often used to reflect this [11]. Authors may also be categorized as a "corresponding author", with contact information like email.
- Relationships between authors and affiliations and/or authors and funding agencies. It is now very common for authors to have multiple affiliations [6] and for multiple authors to share a subset of affiliations or funding agencies. These many-to-many relationships are best encoded as relations rather than repeating the information for each author. These relationships are shown in Figure 1.
- Bibliographic information (e.g., journal or conference name, volume, year, etc.).
- The list of bibliographic references.
- Submission and acceptance dates.
- Licensing information.
- Funding information.

There are numerous other fields that may be encoded into a LaTeX document or the output format produced from LaTeX. Examples include abstract, number of pages, address information for authors, links to ancillary works like code and data, etc. We come from the world of mathematics and computer science, but other things like chemical structures and clinical trials can also be encoded into metadata. It is beyond the scope of this document to catalog all of them, but rather to focus on the most important elements that are common to all academic disciplines.

### 4.1 Metadata schemas

Several organizations have defined schemas for the organization of metadata about an article. One of the most important ones is `crossref.org`, which is a non-profit organization whose primary mission is the collection of metadata and the assignment of DOIs. Their schema supports multiple affiliations, author roles, and funding agencies. Other formats include Elsevier's Scopus indexing service and the Clarivate Web of Science.

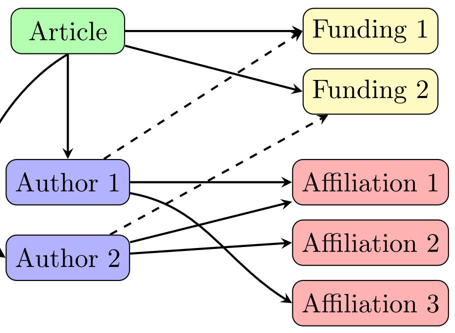


**Figure 1**: Relationships between major entities. Each entity is listed only once in the LaTeX source. An article may have multiple authors who share relationships to affiliations. Funding agencies are related to the article in the crossref schema, so we chose to link them this way. As an alternative, relations shown with dashed arrows can link authors to their funding sources, in much the same way that we relate authors to their affiliations. We chose to use footnotes to clarify the complex relationships between funding agencies and authors or affiliations. Some funding agencies (e.g., [10]) have strict guidelines for how these annotations should be shown in the paper.

Another important schema is the Journal Article Tag Suite (JATS), which is available in three variations for archiving & metadata, publishing, and authoring [9]. The JATS format may be viewed as a complete structural representation for a publication; in many ways comparable to LaTeX but focused even more on semantic structure rather than typesetting or layout. A JATS document consists of several sections, including front matter, body, and back matter. Most metadata occurs in the front matter and back matter.

There are numerous other formats, but these tend to be less descriptive and incomplete. These include the Dublin Core, the Directory of Open Access Journals (DOAJ), the Extensible Metadata Platform (XMP) that is common in PDF, and PRISM. Among all these alternatives, we found the JATS format to be the most expressive and consistent with others.

## 5 Using unique identifiers

Unfortunately, things like human names and institution names are not unique identifiers. The DBLP bibliographic website lists 14 authors in computer science who use the exact name "Thomas Müller", and dozens of others that are similar to this, like Thomas F. Müller. There are multiple institutions that go by the names MIT or USC. In order to perform large scale bibliometric analysis for attribution or duplicate detection, all entities associated with a publication need to be assigned a unique identifier.

Many of the XML schemas such as JATS have embraced the use of unique identifiers. The most notable efforts to assign unique identifiers include:

- DOIs for publications [13],
- ORCIDs for authors [5],
- ROR IDs for research institutions [7],
- Crossref funder registry for funding agencies [2].

Note that in each case where an organization has assigned a unique ID to an entity, there will often be competing organizations with their own ID space. For example, other identifiers for authors have been issued by Clarivate Web of Science, Scopus, SciENcv, Mathematical Reviews, and DBLP.

ROR IDs have coarse granularity, so while there is an identifier for Massachusetts Institute of Technology, they don't distinguish between departments, schools, or programs of the university. By contrast, Mathematical Reviews assigns institution codes at the department level (e.g., 1-SCA-C for the department of computer science at University of Southern California).

A complete list of identifiers associated with scholarly publications is beyond the scope of this document, and we should expect future ID systems to emerge. Because an entity may have multiple IDs from different organizations, we strongly recommend a schema that assigns IDs with a namespace and identifier within that namespace. Thus for example, an organization may have both a Ringgold ID and an ROR ID. Including both can be helpful.

## 6 Output formats

In our processing, LaTeX is not usually read by humans, but is instead converted into another format like PDF or HTML. To the extent possible, it is desirable to embed the metadata into these output formats in a machine-readable way so that the metadata accompanies the consumable document. Unfortunately the standards for doing so are generally lacking in comprehensiveness.

Probably the most important example of this is the XMP standard, whose standard schema does not even provide a way to identify authors by ORCID. Luckily, as the name implies, this format is extensible, and the XML dictionary may use a schema from a variety of namespaces [14]. Springer does this for ORCIDs by defining their own namespace `sn` and encoding authors as a sequence of `(name, orcid)` pairs. Rather than embracing proprietary extensions such as this, we believe that XMP should use the JATS schema to encode authors, affiliations, funding agencies, and bibliographic references. Unfortunately this is not supported by the `hyperxmp` and `pdfx` packages, but the LaTeX team is engaged in a long-term project to improve the production of XMP in PDF [3].

## 7 `\author` considered harmful

We now turn to the problem of how to embed metadata into the original LaTeX source. The original LaTeX definition of `\author` provides little help in capturing author metadata, and is also problematic for displaying large numbers of authors. In the standard `article` class, the author defines `\author` to include blocks of formatted text, separated by `\and`. Thus for example, there is no standard way to associate an ORCID with an author's name, or to associate affiliations or funding agencies with an author. Left to their own devices, authors might use various embedded macros or footnotes to link authors to their metadata, and this makes it very difficult to extract metadata from the LaTeX.

Part of the problem here is that the `\author` macro is intricately woven into the *display* of author information on the page. This is an example where the separation of concerns has been neglected, mixing structure with display. Because of this past history with the `\author` macro, we deliberately chose to break `\author` and use `\addauthor` instead. This means authors have to do some work to convert from other standard LaTeX classes to our class, but we judged that to be necessary because of the bad habits that LaTeX has encouraged.

We are not the first to have recognized the deficiency of `\author`. Some LaTeX styles have improved upon the basic use of `\author`, and have adopted metadata capture as part of their authoring process. Examples include `ltugboat`, `elsarticle`, `acmart`, and `amsart`. Each of these uses some variation on `\author` to capture some metadata about an article, but none of them rise to the level of expressiveness contained in something like JATS. Moreover, we are unaware of any that have attempted to provide functionality for a publishing workflow by extracting the metadata from the LaTeX. Publishing workflows tend to be proprietary, but most use significant human labor that is covered by their business model.

## 8 The `iacrcc` LaTeX and BibTeX styles

Building on what we have learned from previous efforts, we have designed a new document class called `iacrcc`[4] that allows us to capture as much metadata as possible from a document. This may be used with either BibTeX with our own `iacrcc.bst` style, or with the `biblatex` package. These files are designed

[4] May be downloaded from `publish.iacr.org/iacrcc`.

to be used in a publishing workflow to produce metadata in several different formats. Not only do they produce metadata to go back into PDF, but they also produce a plain text version of metadata that can be easily processed for other purposes like DOI registration. We capture a broad range of metadata, including alternate titles, author names, surnames, ORCIDs, affiliations with ROR IDs and addresses, and abstract. An example of author metadata for iacrcc is given in Figure 2.

```
\title[running={Emojex documentation},
       onclick={example.com/emo},
       subtitle={Faces in unicode},
      ]{Emojex: use of emojis in \LaTeX}
\addauthor[orcid={0000-0002-0599-0192},
           inst={1,2},
           onclick={www.madmagazine.com/}
           email={fester@example.com},
          ]{Fester \surname{Bestertester}}
\addauthor[orcid={0000-0001-7890-5430},
           inst={2},
           footnote={Thanks mom!},
          ]{Kevin S. \surname{McCurley}}
\affiliation[ror=044t1p926,
             city={New York},
             country={United States}]{MAD}
\affiliation[country={United States}]{Self}
\addfunding[crossref=100011047,
            grantid={A-1234},
            country={Canada}
           ]{AGE-WELL}
```

**Figure 2**: Sample metadata entry in `iacrcc.cls`.

## 8.1 How it works

The workflow for an author consists of the usual multiple rounds of running `latex`, `bibtex` or `biber`, followed by two more runs of `latex`. The output from this is not only a PDF file with XMP metadata, but also a file `\jobname.meta` file that contains all metadata in a structured format. The `.meta` file is written with macros using `\write` calls.

The structure of the `\jobname.meta` is similar to YAML. We thought about attempting to write YAML or JSON or XML format, but each output format has its own set of special characters and encoding requirements that are complicated to achieve in LaTeX. It was easier for us to write Python code to parse our custom output format than to write LaTeX code to produce one of the more common formats. This Python code is included in the repository for the `iacrcc` files.[5]

[5] See the github repository at `github.com/IACR/latex`.

The basic metadata from the paper is written to the `.meta` file using macros from the `iacrcc.cls` file. The citation information is written into the `.meta` file in one of two different ways, depending on whether the author chooses to use BibTeX or `biblatex`. Both methods produce a `.bbl` file that contains `\write` macros to append to the `.meta` file during compilation. The `\write` macros are implemented in the `iacrcc.cls` file for `biblatex`, and are implemented in the `iacrcc.bst` file for BibTeX. In both cases, the `.bbl` file ends up containing a structured form of the citations. In theory, this allows us to follow the standard practice of publishers to only require authors to submit their `.bbl` file rather than their entire BibTeX file. In practice we require authors to submit their BibTeX because there is no convenient way to validate the `.bbl` file.

## 8.2 The submission pipeline

Once a paper has been accepted for publication, the authors need only submit their LaTeX source file(s), including the BibTeX file they used. The submission form is minimal, since all metadata is included in the LaTeX and BibTeX files themselves. We merely capture an authenticated `paperid` and require the submitting author to supply an email address for the contact author. We derive the DOI from the `paperid`, and inject it into the PDF during compilation along with the acceptance and received dates.

Once the authors upload their LaTeX sources, the server runs `latexmk` within a docker container containing an instance of TeX Live. The server validates that the sources were compiled, and provides reports back to the author in case of any errors. We plan to release our server code as open source in the future, but it's premature to do so now, since some basic design decisions are still being made.

Once the document successfully compiles, the server runs a Python script to process the `.meta` file, creating metadata in XMP, JATS, JSON, and crossref formats. The JSON format is convenient for immediately publishing the article on the web. The crossref format may be used to register the paper with a DOI.

If the author is satisfied with the output from compiling their source, then the paper moves to the next step of copyediting. Copyediting is itself a huge topic in publishing that is mostly beyond the scope of this article. In our experience with external publishers, some of the effort is devoted to metadata handling. Our goal is to at least completely automate metadata handling.

Once the paper is given final approval by the copyeditor, the paper may be published without need for a human to handle any of the metadata. At the time the paper is published, the DOI is registered.

## 9 Summary

We believe that LaTeX can be used to simplify the processing of metadata in the publishing process, and we have developed a document class that we hope will greatly improve the quality of our metadata. By using this approach, we believe it should be possible to streamline the publishing workflow of an open access journal with a low budget. We are in the early stages of this project, and we welcome suggestions for better ways to capture metadata.

Metadata handling is just one reason why text extraction is important for LaTeX. We are in the midst of a revolution in natural language processing through the development of machine learning for large language models. We are hopeful that this will give rise to better tools for tasks such as copy editing. This includes some fairly mechanical steps like punctuation, spelling, and grammar checking. It may also involve visual aspects of typography (e.g., widows, orphans, under/overfull hboxes). It can also involve more intensive steps like checking consistency in terminology, optimizing word choices, or improving sentence structure.

Unfortunately, one barrier to the use of large language models with LaTeX is the fact that it is relatively difficult to extract the author's text from LaTeX. We encourage the community to think more about this problem — not just within author environments or PDF output, but also within publishing pipelines.

### Acknowledgements

The authors would like to thank Gaëtan Leurent and other contributors to the `iacrtrans` document class that was used as the starting point of this project. The authors would also like to thank Enrico Gregorio and David Carlisle for answering questions about the inner workings of LaTeX, and to the reviewers of this paper for making very useful suggestions.

### References

[1] J. Bosman, J.E. Frantsvåg, et al. OA diamond journals study. Part 1: Findings, Mar. 2021. This report was supported by Science Europe and cOAlition S. `doi.org/10.5281/zenodo.4558704`

[2] Crossref funder registry. `crossref.org/services/funder-registry/`

[3] U. Fischer, F. Mittelbach. Adding XMP metadata in LaTeX. *TUGboat* 135(3):263–267, 2022. `doi.org/10.47397/tb/43-3/tb135fischer-xmp`

[4] A. Grossmann, B. Brembs. Current market rates for scholarly publishing services. *F1000Research*, 2021. `doi.org/10.12688/f1000research.27468.2`

[5] L.L. Haak, M. Fenner, et al. ORCID: a system to uniquely identify researchers. *Learned Publishing* 25(4):259–264, 2012.

[6] H. Hottenrott, M.E. Rose, C. Lawson. The rise of multiple institutional affiliations in academia. *Journal of the Association for Information Science and Technology* 72(8):1039–1058, 2021. `doi.org/10.1002/asi.24472`

[7] R. Lammey. Solutions for identification problems: a look at the research organization registry. *Science Editing* 7(1):65–69, 2020.

[8] L. Lamport. *LaTeX: A Document Preparation System.* Addison-Wesley Publishing Company, first ed., 1986.

[9] National Center for Biotechnology Information (NCBI). Journal publishing tag library NISO JATS version 1.3. `jats.nlm.nih.gov/publishing/tag-library/1.3/`, June 2021.

[10] National Institutes of Health. Communicating and acknowledging federal funding, 2021. `grants.nih.gov/policy/federal-funding.htm`

[11] NISO. CRediT: Contributor roles taxonomy. `credit.niso.org`

[12] Open Journal Systems. `pkp.sfu.ca/ojs/`

[13] N. Paskin. Digital object identifier (DOI$^{®}$) system. *Encyclopedia of library and information sciences* 3:1586–1592, 2010.

[14] *Technical Note 0009: XMP Extension Schemas in PDF/A-1.* `www.pdfa.org/resource/technical-note-tn-0009-xmp-extension-schemas-in-pdfa-1/`

◇ Joppe W. Bos
`joppe.bos (at) nxp dot com`
ORCID 0000-0003-1010-8157

◇ Kevin S. McCurley
`iacrcc (at) digicrime dot com`
ORCID 0000-0001-7890-5430

# LaTeX anniversaries — A look in two directions

Frank Mittelbach

Depending on how you count we have several LaTeX anniversaries to celebrate in 2023: roughly forty years ago Leslie Lamport started his work on LaTeX (which became LaTeX 2.09 in 1986). Ten years later in 1993 we made the first beta version of LaTeX $2_\varepsilon$ available — since then the standard LaTeX version used across the world.

Thirty years of LaTeX $2_\varepsilon$ does not mean three decades of standstill — on the contrary. During that time thirty-six new kernel versions have been released and the LaTeX ecosystem grew from a few hundred add-on packages to several thousands.

However, during the first two decades changes to the core of LaTeX were rather minor and most activity was concentrated in the package universe, but the last decade showed an increased level of activity modernizing the LaTeX core functionalities. This started around 2015 when the LaTeX Project Team reimported bug fixes accumulated in a separate package back into the kernel. Since then the format was gradually modernized, e.g., by making UTF-8 the default in 2018 and by incorporating the L3 programming layer in 2020. This intensified further in the last two years when the team embarked on a multi-year journey to enable automatic tagging of the PDF output produced from LaTeX.

Once the results of this project are fully available it will be possible to generate accessible documents with LaTeX without the need to post-process the LaTeX output. With the June 2023 release of LaTeX a major milestone of this project will be reached. With this release a restricted class of documents can already be automatically tagged — the digital version of this article is an example for this.

Together with the first release of LaTeX $2_\varepsilon$ the first edition of *The LaTeX Companion* [12] was published. In 2004 the second edition [42] (describing the extended ecosystem of LaTeX $2_\varepsilon$) hit the streets, and finally, after five years of writing, the third edition [43] has been published as a two-volume set this time — a living testimony to the widespread use of LaTeX and its by now huge ecosystem.

The remainder of this article consists of an excerpt[1] from this third edition of *The LaTeX Companion* that describes the LaTeX history in more detail.

◇ Frank Mittelbach
Mainz, Germany
https://www.latex-project.org

## A brief history (of nearly half a century) — excerpt from *The LaTeX Companion, 3rd edition*

*In the Beginning …*

In May 1977, Donald Knuth of Stanford University [21] started work on the text-processing system that is now known as "TeX and METAFONT" [14–18]. In the foreword of *The TeXbook* [14], Knuth writes: "TeX [is] a new typesetting system intended for the creation of beautiful books — and especially for books that contain a lot of mathematics. By preparing a manuscript in TeX format, you are telling a computer exactly how the manuscript is to be transformed into pages whose typographic quality is comparable to that of the world's finest printers."

In 1979, Gordon Bell wrote in a foreword to an earlier book, *TeX and METAFONT, New Directions in Typesetting* [13]: "Don Knuth's Tau Epsilon Chi (TeX) is potentially the most significant invention in typesetting in this century. It introduces a standard language in computer typography and in terms of importance could rank near the introduction of the Gutenberg press."

In the early 1990s, Donald Knuth produced an updated version and also officially announced that TeX would not undergo any further development [22, 23] in the interest of stability. Perhaps unsurprisingly, the 1990s saw a flowering of experimental projects that extended TeX in various directions; many of these are coming to fruition in the early 21st century, making it an exciting time to be involved in automated typography.

The development of TeX from its birth as one of Don's "personal productivity tools" (created simply to ensure the rapid completion and typographic quality of his then-current work on *The Art of Computer Programming*) [19] was largely influenced and nourished by the American Mathematical Society on behalf of U.S. research mathematicians.

*… and Lamport saw that it was Good.*

While Don was developing TeX, in the early 1980s, Leslie Lamport started work on the document preparation system now called LaTeX, which used TeX's typesetting engine and macro system to implement a declarative document description language based on that of a system called

[1] © 2023, Pearson. Reprinted with permission.

doi.org/10.47397/tb/44-1/tb136mitt-history

Scribe by Brian Reid [50]. The appeal of such a system is that a few high-level LaTeX declarations, or commands, allow the user to easily compose a large range of documents without having to worry much about their typographical appearance. In principle at least, the details of the layout can be left for the document designer to specify elsewhere.

The second edition of *LaTeX: A Document Preparation System* [25] begins as follows: "LaTeX is a system for typesetting documents. Its first widely available version, mysteriously numbered 2.09, appeared in 1985." This release of a stable and well-documented LaTeX led directly to the rapid spread of TeX-based document processing beyond the community of North American mathematicians.

LaTeX was the first widely used language for describing the logical structure of a large range of documents and hence introducing the philosophy of logical design, as used in Scribe. The central tenet of "logical design" is that the author should be concerned only with the logical content of his or her work and not its visual appearance. Back then, LaTeX was described variously as "TeX for the masses" and "Scribe liberated from inflexible formatting control". Its use spread very rapidly during the next decade. By 1994 Leslie could write, "LaTeX is now extremely popular in the scientific and academic communities, and it is used extensively in industry." But that level of ubiquity looks quite small when compared with the present day when it has become, for many professionals on every continent, a workhorse whose presence is as unremarkable and essential as the workstation on which it is used.

*Going global*

The worldwide availability of LaTeX quickly increased international interest in TeX and in its use for typesetting a range of languages. LaTeX 2.09 was (deliberately) not globalized, but it was globalizable; moreover, it came with documentation worth translating because of its clear structure and straightforward style. Two pivotal conferences (Exeter UK, 1988, and Karlsruhe Germany, 1989) established clearly the widespread adoption of LaTeX in Europe and led directly to International LaTeX [54] and to work led by Johannes Braams [1] on more general support for using a wide variety of languages and switching between them (see Chapter 13).

Note that in the context of typography, the word *language* does not refer exclusively to the variety of natural languages and dialects across the universe; it also has a wider meaning. For typography, "language" covers a lot more than just the choice of "characters that make up words", as many important distinctions derive from other cultural differences that affect traditions of written communication. Thus, important typographic differences are not necessarily in line with national groupings but rather arise from different types of documents and distinct publishing communities.

*The Next Generation*

Another important contribution to the reach of LaTeX was the pioneering work of Frank Mittelbach and Rainer Schöpf on a complete replacement for LaTeX's interface to font resources, the New Font Selection Scheme (NFSS) (see Chapter 9). They were also heavily involved in the production of the AMS-LaTeX system that added advanced mathematical typesetting capabilities to LaTeX (see Chapter 11).

As a reward[2] for all their efforts, which included a steady stream of bug reports (and fixes) for Leslie, by 1989 Frank and Rainer "were allowed" to take over the maintenance and further development of LaTeX. One of their first acts was to consolidate International LaTeX as part of the kernel[3] of the system, "according to the standard developed in Europe". Very soon version 2.09 was formally frozen, and although the change-log entries continued for a few months into 1992, plans for its demise as a supported system were already far advanced as something new was badly needed.

*Too much of a Good Thing™*

The worldwide success of LaTeX had by the early 1990s led in a sense to too much development activity: under the hood of Leslie's "family sedan" many TeXnicians had been laboring to add such goodies as super-charged, turbo-injection, multivalved engines and much "look-no-thought" automation. Thus, the announcement in 1994 of the new standard LaTeX, christened LaTeX $2_\varepsilon$, explains its existence in the following way:

> Over the years many extensions have been developed for LaTeX. This is, of course, a sure sign of its continuing popularity but it has had one unfortunate result: incompatible LaTeX formats came into use at different sites. Thus, to process documents from various places, a site maintainer was forced to keep LaTeX (with and without NFSS), SLITeX,

[2] Pronounced "punishment".

[3] *Kernel* here means the core, or center, of the system.

AMS-LaTeX, and so on. In addition, when looking at a source file it was not always clear for which format the document was written.

To put an end to this unsatisfactory situation a new release of LaTeX was produced. It brings all such extensions back under a single format and thus prevents the proliferation of mutually incompatible dialects of LaTeX 2.09.

*Standard LaTeX (LaTeX 2ε)*

The development of this "New Standard LaTeX" and its maintenance system was started in 1993 by the LaTeX Project Team [45], which soon comprised the author of this book, Rainer Schöpf, Chris Rowley, Johannes Braams, Michael Downes, David Carlisle, Alan Jeffrey, and Denys Duchier, with some encouragement and gentle bullying from Leslie. Although the major changes to the basic LaTeX system (the kernel) and the standard document classes (styles in 2.09) were completed by 1994, substantial extra support for colored typography, generic graphics, and fine positioning control were added later, largely by David Carlisle. Access to fonts for the new system incorporated work by Mark Purtill on extensions of NFSS to better support variable font encodings and scalable fonts [2–4].

*1994 — The first edition of the LaTeX Companion*

At this point in the story the first edition of the *LaTeX Companion* was written, which helped a lot in making many important packages known to a wide audience and as a side effect helped shape a standard corpus of LaTeX packages expected to be available on any installation across the world.

*Towards the 21st century*

Although the original goal for this LaTeX 2ε was consolidation of the wide range of incompatible models carrying the LaTeX marquee, what emerged was a substantially more powerful system with both a robust mechanism (via LaTeX packages) for extension and, importantly, a solid technical support and maintenance system. This provides robustness via standardization and maintainability of both the code base and the support systems. The core of this system remains the current standard LaTeX system that is described in this book. It has fulfilled most of the goals for "a new LaTeX for the 21st Century", as they were envisaged back in 1989 [48, 49].

The specific claims of the current system are "… better support for fonts, graphics and color; actively maintained by the LaTeX Project Team". The details of how these goals were achieved, and the resulting subsystems that enabled the claims to be substantially attained, form a revealing study in distributed software support: the core work was done in at least five countries and, as is illustrated by the bugs database [27], the total number of active contributors to the technical support effort remains high.

*The package system*

Although the LaTeX kernel suffered a little from feature creep in the late 1990s, the package system together with the clear development guidelines and the legal framework of the LaTeX Project Public License (LPPL) [29, 34] have enabled LaTeX to remain almost completely stable while supporting a wide range of extensions. These have largely been provided by a similarly wide range of people who have, as the project team are happy to acknowledge and the online catalogue [56] bears witness, enhanced the available functionality in a vast panoply of areas.

*Development work*

All major developments of the base system have been listed in the regular issues of *LaTeX News* [26]. At the turn of the century, development work by the LaTeX Project Team focused on the following areas: supporting multi-language documents [32]; a "Designer Interface for LaTeX" [40]; major enhancements to the output routine [33]; improved handling of inter-paragraph formatting; and the complex front-matter requirements of journal articles. Back then prototype code had been made available (see [39]), but the work has otherwise been kept separate from LaTeX — partly because it was executing simply too slowly on the available hardware.

*No new features at the kernel level …*

One thing the project team steadfastly refused to do at that time was to unnecessarily "enhance" the kernel by providing additional features as part of it, thereby avoiding the trap into which LaTeX 2.09 fell in the early 1990s: the disintegration into incompatible dialects where documents written at one site could not be successfully processed at another site. In this discussion it should not be forgotten that LaTeX serves not only to produce high-quality documents but also to enable collaboration and exchange by providing a lingua franca for various research communities.

With LaTeX 2ε, documents written in 1996[4] can still be run with today's LaTeX. In the opposite direction, new documents run on older kernel releases if the additional packages used are brought

[4] The time between 1994 and 1996 was a consolidation time for LaTeX 2ε, with major fixes and enhancements being made until the system was thoroughly stable. In fact, with some minor alterations in pagination or font usage, it is usually possible to reprocess even documents from the eighties (i.e., written for LaTeX 2.09) or make them reusable with little effort.

up-to-date — a task that, in contrast to updating the LaTeX kernel software, is easily manageable even for users working in a multiuser environment (e.g., in a university or company setting).

*… but no standstill*

But a stable kernel is not identical to a standstill in software development; of equally crucial importance to the continuing relevance and popularity of LaTeX is the diverse collection of contributed packages building on this stable base. The success of the package system for nonkernel extensions is demonstrated by the enthusiasm of these contributors — many thanks to all of them! As can be easily appreciated by visiting the highly accessible and stable Comprehensive TeX Archive Network (see Appendix C) or by reading this book (where more than 250 of these "Good Guys"[5] are listed on page II-967), this has supported the existence of an enormous treasure trove of LaTeX packages and related software.

*The back office*

The provision of services, tools, and systems-level support for such a highly distributed maintenance and development system was itself a major intellectual challenge, because many standard working methods and software tools for these tasks assume that your colleagues are in the next room, not the next continent (and in the early days of the development, e-mail and FTP were the only reliable means of communication). The technical inventiveness and the personalities of everyone involved were both essential to creating this example of the friendly face of open software maintenance, but Alan Jeffrey and Rainer Schöpf deserve special mention for "fixing everything".

A vital part of this system that is barely visible to most people is the regression testing system with its vast suite of test files [31]. It was initially devised and set up by Frank and Rainer with Daniel Flipo; it has proved its worth countless times in the never-ending battle with the bugs. Over the years it has seen many refinements, cumulating in a complete rewrite as part of `l3build` [44], which we describe in Section 17.3 on page II-606.

*2004 — The second edition of the LaTeX Companion*

In 2004, i.e., roughly a decade after its first edition, the second edition of the *LaTeX Companion* was published. Due to the popularity of LaTeX $2_\varepsilon$ and its extended features for developers, new important packages had emerged, and LaTeX had reached out into new domains. While the advice given in the first edition remained largely valid (last but not least because of the long-term backward compatibility paradigm of LaTeX), we ended up rewriting 90% of the original content and added about 600 pages to account for new developments. As before, the second edition helped a lot in standardizing the use, and this way the interoperability, of LaTeX across the world.

*Research*

Some members of the LaTeX Project Team have built on the team's experience to extend their individual research work in document science beyond the current LaTeX structures and paradigms. Some examples of their work up to now can be found in the following references: [5, 7–9, 35–38, 46, 51, 53]. An important spin-off from the research work was the provision of some interfaces and extensions that are immediately usable with standard LaTeX.

*…and into the future*

The decision to keep the core of the standard LaTeX system stable and essentially unchanging had two major advantages over any other approach to support fully automated document processing. First, the system already efficiently provided high-quality formatting of a large range of elements in very complex documents of arbitrary size. Second, it was robust in both use and maintenance and hence offered the potential to remain in widespread use for at least a further 15 years.[6] In the second edition of this book we wrote on this topic:

> As more such functionality is added, it will become necessary to assess the likelihood that merely extending LaTeX in this way will provide a more powerful, yet still robust and maintainable, system. This is not the place to speculate further about the future of LaTeX but we can be sure that it will continue to develop and to expand its areas of influence whether in traditional publishing or in electronic systems for education and commerce.

*Reassessment time*

This reassessment became necessary in the second decade of the new century, when it became obvious that this position was gradually getting unsustainable, because more and more areas in which people were looking for solutions could not be adequately addressed with a model of a fixed

[5] Unfortunately, this is nearly the literal truth: you need a keen eye to spot the few ladies listed.

[6] One of the authors of the second edition had publicly staked a modest amount of beer on TeX remaining in general use (at least by mathematicians) until at least 2010. He should have made a larger bet, given that this is now 2022 and LaTeX is healthy and in fact growing its user base due to its many unsurpassed qualities.

kernel and all developments outsourced to the package level. Examples are the move to Unicode in basically all operating systems and the growing pressure to produce "accessible" documents that conform to standards such as PDF/UA (Portable Document Format/Universal Accessibility).

*An important policy change*

Thus, in 2015, the LaTeX Project Team changed its policy and restarted kernel development. To retain the best of both worlds this was accompanied by developing a rollback/roll-forward functionality for the kernel and packages (that care to implement it). This allows a current LaTeX format to roll back to an earlier point in time in order to process old documents that rely on interfaces that have been changed since then or to process documents that explicitly worked around bugs (and so expect them to be there) that have been fixed in the meantime.

The first action of the team was to retire the fixltx2e package and instead include the accumulated fixes it contained directly in the format and to officially support LaTeX when using the Unicode engines XeTeX and LuaTeX. A big step forward happened in 2018 when LaTeX switched its default input encoding to UTF-8. This change proved that the policy change was the right thing to do and that the preparatory work (e.g., providing rollback) allows executing even major changes without disruption in its user base in order to keep LaTeX relevant and useful. A good indicator for the renewed and increased activity are the regular LaTeX newsletters [26] accompanying each release, which grew bulkier and again appeared semi-annually.

*And where is the mythical LaTeX3?*

The event of providing the mythical LaTeX3 had long become a standing joke as "two years from 'now' — with 'now' a moving target". The reason was that the concepts and ideas for LaTeX3 have been simply a decade or more too early, and while the team implemented a fully working version already in 1990, it was simply too slow to be usable with the then available computing power. Thus, we gave up pursuing it and instead concentrated on offering LaTeX $2_\varepsilon$, which then went public in 1994.

But ideas and concepts were never forgotten by the team, and especially its newer members (who joined in this century) pushed them back to the forefront and improved them dramatically. As a result, the code was eventually publicly made available as the expl3 package. It was then picked up by a number of enthusiastic package developers and used as the basis for their new packages. For example, if you use acro, breqn, fontspec, siunitx, unicode-math, or xparse, to name a few, you use "LaTeX3" under the hood; a recent count shows more than 200 such packages or classes as part of TeX Live.

So in 2019 the LaTeX Project Team made two wide-ranging decisions: there will not be a separate LaTeX3 that is being developed alongside LaTeX $2_\varepsilon$ (as was originally planned). Instead, we will modernize the current LaTeX gradually from the inside, using the new rollback mechanism and "development" formats as a safety net to ensure that there is no disruption of service for our user base. As a first step on this journey, the L3 programming layer and the LaTeX3 document-level command declarations (formerly known as expl3 and xparse) were made an integral part of LaTeX on February 2, 2020. Thus, more or less exactly 30 years after its conception, LaTeX3 became a reality for every LaTeX user — even though few will have immediately noticed.

*. . . well it got merged into the kernel in 2020*

*The foundation layer for modernization*

The importance of this step is that it allows the team to modernize other parts of the kernel and develop new functionality entirely based on the L3 programming layer, which offers many features not available with legacy LaTeX programming constructs. For example, the new Hook Management System for LaTeX, which is a cornerstone for modernizing and transforming the existing LaTeX, is entirely written using the new L3 programming layer, and other parts will follow suit.

*Today's challenge: structured and accessible output is needed*

As already mentioned, there is a steadily increasing interest in the production of "tagged" PDF documents that are "accessible", in the sense that they contain information to assist screen reading software, etc., and, more formally, that they adhere to the PDF/UA (Portable Document Format/Universal Accessibility) standard [55], explained further in [10]. In many disciplines this is starting to become a requirement when applying for grants or when publishing results.

At the moment, all methods of producing such "accessible PDFs", including the use of LaTeX, require extensive manual labor in preparing the source or in post-processing the PDF (maybe even at both stages); and these labors often have to be repeated after making even minimal changes to the (LaTeX or other) source. This is a huge pity, because LaTeX should in theory be well-positioned to do this work automatically, given that its source is already well-structured.

The production of tagged (i.e., structured) PDF documents is not only important in order to comply to accessibility standards. It also opens possibilities to reuse data from such PDFs, because it allows other applications to correctly identify the structure inside the output document and this way extract or manipulate parts of the content — workflows that become increasingly important in the digital world.

The LaTeX Project Team has for some years been well aware that these new usages are not adequately supported by the current system architecture of LaTeX $2_\varepsilon$ and that major work in this area is therefore urgently needed to ensure that LaTeX remains an important and relevant document source format. However, the amount of work required to make such major changes to the LaTeX system architecture is enormous and definitely way beyond the limited resources of a small team of volunteers working in their spare time (or maybe just about possible, but only given a very long — and most likely too long — period of time).

*A multi-year project to shape the future of LaTeX*

At the TeX Users Group conference 2019 in Palo Alto the team's previously pessimistic outlook on this subject became cautiously optimistic, because of discussions with senior executives from Adobe about the possibility of producing structured PDF from LaTeX source without the need for the usual requirement of considerable manual post-processing. As a result of these discussions, towards the end of 2019 the team produced an extended feasibility study for the project, aimed primarily at Adobe engineers and decision-makers. This study [41] describes in some detail the various tasks that constitute the project and their interdependencies. It also contains a project plan covering how, and in what order, these tasks should be tackled both to achieve the final goal and, at the same time, to provide intermediate concrete results that are relevant to user communities (both LaTeX and PDF); these intermediate results will help in obtaining feedback that is essential to the successful completion of later tasks.

This multi-year project found the approval of Adobe, which then committed to financially and otherwise supporting this endeavor [47]. Unfortunately — thanks to the COVID-19 pandemic — the start got delayed, but since the end of 2020, this exciting project is now well under way. First results from this project that are already in existence (such as the new hook management system and the alignment of the hyperref package with the LaTeX kernel) are already described in this book. Other parts are obviously still vaporware at this point. Fortunately, none is expected to render any documentation or suggestion made in this book obsolete — after all, the project goal is to enable tagging of existing documents, simply by reprocessing with minor configuration changes as outlined in the "Spoiler alert" Section 2.1.1 on page 23.

## References

[1] Johannes Braams. "Babel, a multilingual style-option system for use with LaTeX's standard document styles". *TUGboat*, 12(2):291–301, 1991.
The babel package was originally a collection of document-style options to support different languages. An update was published in *TUGboat*, 14(1):60–62, April 1993. https://tug.org/TUGboat/tb12-2/tb32braa.pdf https://tug.org/TUGboat/tb14-1/tb38braa.pdf

[2] David Carlisle. "A LaTeX tour, Part 1: The basic distribution". *TUGboat*, 17(1):67–73, 1996.
A "guided tour" around the files in the basic LaTeX distribution. File names and paths relate to the file hierarchy of the CTAN archives. https://tug.org/TUGboat/tb17-1/tb50carl.pdf

[3] ——. "A LaTeX tour, Part 2: The tools and graphics distributions". *TUGboat*, 17(3):321–326, 1996.
A "guided tour" around the "tools" and "graphics" packages. Note that Lamport's manual [25] assumes that at least the graphics distribution is available with standard LaTeX. https://tug.org/TUGboat/tb17-3/tb52carl.pdf

[4] ——. "A LaTeX tour, Part 3: mfnfss, psnfss and babel". *TUGboat*, 18(1):48–55, 1997.
A "guided tour" through three more distributions that are part of the standard LaTeX system. The mfnfss distribution provides LaTeX support for some popular METAFONT-produced fonts that do not otherwise have any LaTeX interface. The psnfss distribution consists of LaTeX packages giving access to PostScript fonts. The babel distribution provides LaTeX with multilingual capabilities. https://tug.org/TUGboat/tb18-1/tb54carl.pdf

[5] ——. "XMLTEX: A non validating (and not 100% conforming) namespace aware XML parser implemented in TeX". *TUGboat*, 21(3):193–199, 2000.
XMLTEX is a an XML parser and typesetter implemented in TeX, which by default uses the LaTeX kernel to provide typesetting functionality. https://tug.org/TUGboat/tb21-3/tb68carl.pdf

[6] David Carlisle, editor. Mathematical Markup Language (MathML) Version 4.0. W3C, 1st edition, 2023.
This is the draft specification for a new version of the Mathematical Markup Language; the current version is 3.0 [7]. MathML4 extensions primarily relate to improving accessibility, with new attributes for improving audio rendering. https://www.w3.org/TR/mathml4/

[7] David Carlisle, Patrick Ion, and Robert Miner, editors. Mathematical Markup Language (MathML) Version 3.0. W3C, 2nd edition, 2014.
This is the current specification defining the Mathematical Markup Language; the upcoming version will be [6]. MathML is an XML vocabulary for mathematics, designed for use in browsers and as a communication language between computer algebra systems. The goal of MathML is to enable mathematics to be served, received, and processed on the World Wide Web, just as HTML has enabled this functionality for text. https://www.w3.org/TR/MathML3/

[8] David Carlisle, Patrick Ion, Robert Miner, and Nico Poppelier, editors. Mathematical Markup Language (MathML) Version 2.0. W3C, 2nd edition, 2003.
This is the previous version of the MathML standard [7]. https://www.w3.org/TR/MathML2/

[9] David Carlisle, Chris Rowley, and Frank Mittelbach. "The LaTeX3 Programming Language—a proposed system for TeX macro programming". *TUGboat*, 18(4):303–308, 1997.
Initial proposals for a radically new syntax and software tools. Most of them are now part of the LaTeX format as the L3 programming layer. https://tug.org/TUGboat/tb18-4/tb57rowl.pdf

[10] Olaf Drümmer and Bettina Chang. PDF/UA in a Nutshell — Accessible documents with PDF. PDF Association, 2013.
A nice introduction to the ISO standard 14289-1 for universal accessibility, also known as PDF/UA [55]. It provides key facts, e.g., the requirements of the standard, the current legal situation, etc. https://pdfa.org/resource/pdfua-in-a-nutshell/

[11] Victor Eijkhout. TeX by Topic, A TeXnician's Reference. Lehmanns Media, Berlin, 2014. ISBN 978-3-86541-590-5. Reprint with corrections. Initially published in 1991 by Addison-Wesley. Also available free of charge from the author in PDF format.
A systematic reference manual for the experienced TeX user. The book offers a comprehensive treatment of every aspect of TeX (not LaTeX!), with detailed explanations of the mechanisms underlying TeX's working, as well as numerous examples of TeX programming techniques. https://eijkhout.net/tex/tex-by-topic.html

[12] Michel Goossens, Frank Mittelbach, and Alexander Samarin. The LaTeX Companion. Tools and Techniques for Computer Typesetting. Addison-Wesley, Reading, MA, USA, 1994. ISBN 0-201-54199-8.
The first edition of this book. The second edition [42] was published ten years later in 2004 and the third edition [43] in 2023.

[13] Donald E. Knuth. TeX and METAFONT — New Directions in Typesetting. Digital Press, Bedford, MA, USA, 1979. ISBN 0-932376-02-9.
Contains an article on "Mathematical Typography", describing the author's motivation for starting to work on TeX and the early history of computer typesetting. Describes early (now obsolete) versions of TeX and METAFONT.

[14] ——. The TeXbook, volume A of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986. ISBN 0-201-13447-0. Jubilee 2021 edition, twenty-fifth printing with corrections.
The definitive user's guide and complete reference manual for TeX. A good secondary reading, covering the same grounds, is [11].

[15] ——. TeX: The Program, volume B of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986. ISBN 0-201-13437-3. Jubilee 2021 edition, thirteenth printing with corrections.
The complete source code for the TeX program, typeset with several indices.

[16] ——. The METAFONTbook, volume C of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986. ISBN 0-201-13445-4 (hardcover), 0-201-13444-6 (paperback). Jubilee 2021 edition, twelfth printing with corrections.
The user's guide and reference manual for METAFONT, the companion program to TeX for designing fonts.

[17] ——. METAFONT: The Program, volume D of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986. ISBN 0-201-13438-1. Jubilee 2021 edition, eleventh printing with corrections.
The complete source code listing of the METAFONT program.

[18] ——. Computer Modern Typefaces, volume E of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986. ISBN 0-201-13446-2. Jubilee 2021 edition, eleventh printing with corrections.
More than 500 Greek and Roman letterforms, together with punctuation marks, numerals, and many mathematical symbols, are graphically depicted. The METAFONT code to generate each glyph is given and it is explained how, by changing the parameters in the METAFONT code, all characters in the Computer Modern family of typefaces can be obtained.

[19] ——. The Art of Computer Programming, volumes 1– 4A and Fascicles 5–6. Addison-Wesley, Reading, MA, USA, 1998–2019. ISBN 0-201-89683-4, 0-201-03822-6, 0-201-03803-X, 0-201-03804-8, 0-13-467179-1, and 0-13-439760-6.
Donald Knuth's major work on algorithms and data structures for efficient programming.

[20] ——. Digital Typography. CSLI Publications, Stanford, CA, USA, 1999. ISBN 1-57586-011-2 (cloth), 1-57586-010-4 (paperback).

A comprehensive collection of Knuth's writings on TEX and typography. While many articles in this collection are available separately on the Web, not all of them are, and having them all in one place for studying is an additional benefit.

[21] ——. "Computers and typesetting". In Knuth [20], pp. 555–562.

Remarks presented by Knuth at the Computer Museum, Boston, Massachusetts, on 21 May 1986, at the "coming-out" party to celebrate the completion of TEX. Originally published as: `https://tug.org/TUGboat/tb07-2/tb14knut.pdf`

[22] ——. "The new versions of TEX and METAFONT". In Knuth [20], pp. 563–570.

Knuth explains how he was convinced at the TUG Meeting at Stanford in 1989 to make one further set of changes to TEX and METAFONT to extend these programs to support 8-bit character sets. He goes on to describe the various changes he introduced to implement this feature, as well as a few other improvements. Originally published as: `https://tug.org/TUGboat/tb10-3/tb25knut.pdf`

[23] ——. "The future of TEX and METAFONT". In Knuth [20], pp. 571–572.

In this article Knuth announces that his work on TEX, METAFONT, and Computer Modern has "come to an end" and that he will make further changes only to correct extremely serious bugs. Originally published as: `https://tug.org/TUGboat/tb11-4/tb30knut.pdf`

[24] Donald E. Knuth and Michael F. Plass. "Breaking paragraphs into lines". In Knuth [20], pp. 67–155.

This article, originally published in 1981, addresses the problem of dividing the text of a paragraph into lines of approximately equal length. The basic algorithm considers the paragraph as a whole and introduces the (now well-known TEX) concepts of "boxes", "glue", and "penalties" to find optimal breakpoints for the lines. The paper describes the dynamic programming technique used to implement the algorithm.

[25] Leslie Lamport. LATEX: A Document Preparation System: User's Guide and Reference Manual. Addison-Wesley, Reading, MA, USA, 2nd edition, 1994. ISBN 0-201-52983-1. Reprinted with corrections in 1996.

The ultimate reference for basic user-level LATEX by the creator of LATEX 2.09. It complements the material presented in this book.

[26] LATEX Project Team. "LATEX news".

An issue of *LATEX News* is released with each LATEX 2$\varepsilon$ release, highlighting changes since the last release. There is also a document combining all issues since 1994, which offers a good overview about the history of LATEX 2$\varepsilon$ as well as providing an easy way to find information on all major updates and extensions that have been implemented over the years. Locally available via: `texdoc ltnews`

[27] ——. "Bugs in LATEX software". Website.

The bug reporting and tracking service run by the LATEX team as part of the LATEX 2$\varepsilon$ maintenance activity. `https://www.latex-project.org/bugs/`

[28] ———. The LATEX3 Interfaces, 2023.

The reference manual for the L3 programming layer, which has been part of the LATEX format since 2020 and thus available for package development — the way for LATEX coding going forward. Locally available via: `texdoc interface3`

[29] ——. "The LATEX project public license (version 1.3c)", 2008.

The Open Source License used by the core LATEX 2$\varepsilon$ distribution and many contributed packages. See [34] for background and history. `https://www.latex-project.org/lppl/`

[30] Frank Mittelbach. "E-TEX: Guidelines for future TEX Extensions". *TUGboat*, 11(3):337–345, 1990.

The output of TEX is compared with that of hand-typeset documents. It is shown that many important concepts of high-quality typesetting are not supported and that further research to design a "successor" typesetting system to TEX should be undertaken. A review of the findings, 23 years later, is provided in [35]. `https://tug.org/TUGboat/tb11-3/tb29mitt.pdf`

[31] ——. "A regression test suite for LATEX 2$\varepsilon$". *TUGboat*, 18(4):309–311, 1997.

Description of the concepts and implementation of the test suite used to test for unexpected side effects after changes to the LATEX kernel. One of the most valuable maintenance tools for keeping LATEX 2$\varepsilon$ stable. `https://tug.org/TUGboat/tb18-4/tb57mitt.pdf`

[32] ——. "Language Information in Structured Documents: Markup and rendering—Concepts and problems". In "International Symposium on Multilingual Information Processing", pp. 93–104. Tsukuba, Japan, 1997. Invited paper. Slightly extended in *TUGboat* 18(3):199–205, 1997.

This paper discusses the structure and processing of multilingual documents, both at a general level and in relation to a proposed extension to standard LATEX. `https://tug.org/TUGboat/tb18-3/tb56lang.pdf`

[33] ——. "Formatting documents with floats: A new algorithm for LATEX 2$\varepsilon$". *TUGboat*, 21(3):278–290, 2000.

Descriptions of features and concepts of a new output routine for LATEX that can handle spanning floats in multicolumn page design. `https://tug.org/TUGboat/tb21-3/tb68mittel.pdf`

[34] ——. "Reflections on the history of the LATEX Project Public License (LPPL) — A software license for LATEX and more". *TUGboat*, 32(1):83–94, 2011.

A review of the evolution of LATEX world's predominant license [29]. `https://tug.org/TUGboat/tb32-1/tb100mitt.pdf`

[35] ——. "E-TEX: Guidelines for future TEX Extensions — revisited". *TUGboat*, 34(1):47–63, 2013.
This article compares the output of TEX with that of hand-typeset documents. This is a reassessment of the findings made 23 years earlier [30]. With the new engines the situation has improved, but even though there is now engine support for most problems, the majority of them still represent important and open research problems for high-quality automated typesetting.
https://tug.org/TUGboat/tb34-1/tb106mitt.pdf

[36] ——. "A general framework for globally optimized pagination". In "Proceedings of the 2016 ACM Symposium on Document Engineering", DocEng'16, pp. 11–20. Association for Computing Machinery, New York, NY, USA, 2016. ISBN 978-1-4503-4438-8.
This paper presents research results for globally optimized pagination using dynamic programming and discusses its theoretical background. It was awarded the "ACM Best Paper Award" at the DocEng 2016 conference. A greatly expanded version of this paper (37 pages) titled "A General LuaTEX Framework for Globally Optimized Pagination" was submitted to the Computational Intelligence (Wiley) in 2017 and accepted January 2018 [38]. https://www.latex-project.org/publications/indexbyyear/2016/

[37] ——. "Effective floating strategies". In "Proceedings of the 2017 ACM Symposium on Document Engineering", DocEng'17, pp. 29–38. Association for Computing Machinery, New York, NY, USA, 2017. ISBN 978-1-4503-4689-4.
This paper presents an extension to the general framework for globally optimized pagination described [36]. The extended algorithm supports automatic placement of floats as part of the optimization using a flexible constraint model that allows for the implementation of typical typographic rules. https://www.latex-project.org/publications/indexbyyear/2017/

[38] ——. "A general LuaTEX framework for globally optimized pagination". *Computational Intelligence*, 35(2):242–284, 2019.
This article is an extended version (37 pages) of the 2016 ACM article "A General Framework for Globally Optimized Pagination" [36], providing much more detail and additional research results. The peer-reviewed publication is now freely available.
https://www.latex-project.org/publications/indexbyyear/2020/

[39] Frank Mittelbach, David Carlisle, and Chris Rowley. "Experimental LATEX code for class design". Vancouver, 1999.
At the TEX Users Group conference in Vancouver the LATEX project team gave a talk on models for user-level interfaces and designer-level interfaces in LATEX3 [40]. Most of these ideas have been implemented in prototype implementations (e.g., template design, front matter handling, output routine, galley and paragraph formatting). The source code is documented and contains further explanations and examples; see also [33]. The underlying programming interfaces are since 2020 part of the LATEX format as the L3 programming layer [28].
Articles: https://latex-project.org/publications/indexbytopic/l3-expl3
Code: https://github.com/latex3/latex3

[40] ——. "New interfaces for LATEX class design, Parts I and II". *TUGboat*, 20(3):214–216, 1999.
Some proposals for the first-ever interface to setting up and coding LATEX classes. While all of them were implemented as experimental prototypes (see [39]), they have been developed at a time were computers were not powerful enough to enable them for general use. This has finally changed and several of these ideas are now making their reappearance as part of the "LATEX Tagged PDF" project [47].
https://tug.org/TUGboat/tb20-3/tb64carl.pdf

[41] Frank Mittelbach, Ulrike Fischer, and Chris Rowley. LATEX Tagged PDF Feasibility Evaluation. LATEX Project, 2020.
This is the feasibility study undertaken by the LATEX team prior to initiating the multiyear project for automatically providing tagged PDF with LATEX. It explains in detail both the project goals and the tasks that need to be undertaken and concludes with a detailed project plan. See also [47]. https://latex-project.org/publications/indexbytopic/pdf/

[42] Frank Mittelbach, Michel Goossens, Johannes Braams, David Carlisle, and Chris Rowley. The LATEX Companion. Tools and Techniques for Computer Typesetting. Addison-Wesley, Reading, MA, USA, 2nd edition, 2004. ISBN 0-201-36299-6.
The second edition of this book. The contributing authors have changed over the years.

[43] Frank Mittelbach with Ulrike Fischer. The LATEX Companion, Parts I & II. Tools and Techniques for Computer Typesetting. Addison-Wesley, Reading, MA, USA, 3nd edition, 2023. ISBN 978-0-13-816648-9.
The third edition of this book, published as two-volume set. It is also available in digital formats.
https://www.informit.com/store/latex-companion-parts-i-ii-3rd-edition-9780138166489

[44] Frank Mittelbach, Will Robertson, and LATEX3 team. "l3build — A modern Lua test suite for TEX programming". *TUGboat*, 35(3):287–293, 2014.
The workflow environment used by the LATEX Project Team and others. Supports concepts developed over the years including regression testing methods, distribution builds, uploads to CTAN, and installation support.
https://tug.org/TUGboat/tb35-3/tb111mitt-l3build.pdf
Locally available program documentation: texdoc l3build

[45] Frank Mittelbach and Chris Rowley. "LATEX 2.09 ↪ LATEX3". *TUGboat*, 13(1):96–101, 1992.
A brief sketch of the LATEX3 Project, retracing its history and describing the structure of the system. An update appeared in *TUGboat*, 13(3):390–391, October 1992. A call for volunteers to help in the development of LATEX3 and a list of the various tasks appeared in *TUGboat*, 13(4):510–515, December 1992. Now mainly of historical interest. https://tug.org/TUGboat/tb13-1/tb34mittl3.pdf

[46] ——. "The pursuit of quality: How can automated typesetting achieve the highest standards of craft typography?" In C. Vanoirbeek and G. Coray, editors, "EP92 — Proceedings of Electronic Publishing '92, International Conference on Electronic Publishing, Document Manipulation, and Typography, Swiss Federal Institute of Technology, Lausanne, Switzerland, April 7–10, 1992", pp. 261–273. Cambridge University Press, New York, 1992. ISBN 0-521-43277-4.
This paper compares high-quality craft typography with the state of the art in automated typesetting. It explains why the current paradigms of computerized typesetting will not serve for high-quality formatting and suggests directions for the further research necessary to improve the quality of computer-generated layout.

[47] ——. "LaTeX Tagged PDF — a blueprint for a large project". *TUGboat*, 41(3):292–298, 2020.
An introduction and summary of the extended feasibility study [41] for the multiyear project "LaTeX Tagged PDF".
`https://latex-project.org/publications/indexbytopic/pdf/`

[48] Frank Mittelbach and Rainer Schöpf. "With LaTeX into the nineties". *TUGboat*, 10(4):681–690, 1989.
This article proposes a reimplementation of LaTeX that preserves the essential features of the current interface while taking into account the increasing needs of the various user communities. It also formulates some ideas for further developments. It was instrumental in the move from LaTeX 2.09 to LaTeX 2$\varepsilon$. `https://tug.org/TUGboat/tb10-4/tb26mitt.pdf`

[49] ——. "Towards LaTeX 3.0". *TUGboat*, 12(1):74–79, 1991.
The objectives of the LaTeX3 project are described. The authors examine enhancements to LaTeX's user and style file interfaces that are necessary to keep pace with modern developments, such as SGML. They also review some internal concepts that need revision.
`https://tug.org/TUGboat/tb12-1/tb31mitt.pdf`

[50] Brian Reid. Scribe: A Document Specification Language and its Compiler. Ph.D. thesis, Carnegie-Mellon University, Pittsburgh, PA 15213, 1980.
The Ph.D. thesis that was one of the inspirations for LaTeX.
`http://reports-archive.adm.cs.cmu.edu/anon/scan/CMU-CS-81-100.pdf`

[51] Chris Rowley. "Models and languages for formatted documents". *TUGboat*, 20(3):189–195, 1999.
Explores many ideas around the nature of document formatting and how these can be modeled and implemented.
`https://tug.org/TUGboat/tb20-3/tb64rowl.pdf`

[52] ——. "The LaTeX legacy: 2.09 and all that". In PODC'01: "Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing 2001, Newport, Rhode Island, United States", pp. 17–25. ACM Press, New York, NY, USA, 2001. ISBN 1-58113-383-9.
Part of a celebration for Leslie Lamport's sixtieth birthday; a very particular account of the technical history and philosophy of TeX and LaTeX. `https://www.latex-project.org/publications/indexbytopic/2e-concepts`

[53] Chris Rowley and Frank Mittelbach. "Application-independent representation of multilingual text". In "Europe, Software + the Internet: Going Global with Unicode: Tenth International Unicode Conference, March 10–12, 1997, Mainz, Germany", The Unicode Consortium, San Jose, CA, 1997.
Explores the nature of text representation in computer files and the needs of a wide range of text-processing software.
`https://latex-project.org/publications/1996-FMi-CAR-UnicodeConf-appl-independent-representation.pdf`

[54] Joachim Schrod. "International LaTeX is ready to use". *TUGboat*, 11(1):87–90, 1990.
Announces some of the early standards for globalization work on LaTeX. `https://tug.org/TUGboat/tb11-1/tb27schrod.pdf`

[55] Technical Committee ISO/TC 171/SC 2. ISO 14289-1:2014 Document management applications — Electronic document file format enhancement for accessibility — 1: Use of ISO 32000-1 (PDF/UA-1), 2014.
ISO 14289-1:2014 specifies the use of the ISO 32000-1:2008 standard to produce accessible electronic documents.
`https://iso.org/standard/64599.html`

[56] Graham Williams. "Graham Williams' TeX Catalogue". *TUGboat*, 21(1):17–90, 2000.
In 2000 this catalogue listed more than 1500 TeX, LaTeX, and related packages and tools on 74 pages and was linked directly to the items on CTAN. CTAN now offers it in the form of several indexes with more than 5000 items covering everything stored there.
`https://tug.org/TUGboat/tb21-1/tb66catal.pdf`
Latest version on CTAN at: `https://ctan.org/pkg/catalogue`

# An introduction to expl3

Marei Peischl

Even some long-term LaTeX users seem to be scared of expl3 — the syntax of the LaTeX3 programming layer — and think of the structure as confusing or even frightening. Perhaps with some justification:

```
\ExplSyntaxOn
\clist_map_inline:nn
    \l_tmpa_clist
    { \__ptxcd_add_item:n {#1} }
\ExplSyntaxOff
```

LaTeX3 is no longer a development for which LaTeX users have been waiting for decades. LaTeX3 has been around for a long time and nowadays is used by all LaTeX users, often without being noticed. The goal of this tutorial is to demystify expl3.

The LaTeX3 programming layer is the foundation of almost all new LaTeX development in the last years. It provides unified interfaces that can be used directly or indirectly by package authors and users to code complex mechanisms or process content much more flexibly than with classic LaTeX.

Overall, the most important goals of LaTeX3 are:

- Uniform interfaces for functions and variables
- Modernization of syntax
- Simplification of controlling expansion

and thus provide both much simpler and more powerful ways to program in LaTeX [5].

Programming is useful when a document, depending on settings, should get either a different layout or a different structure. A typical example from teaching is the creation of an exam including solutions within a single file, where solutions can be hidden. Another use case is the processing of external data. A typical example is a list to be converted to an enumeration:

```
\ExplSyntaxOn
\begin{enumerate}
    \clist_map_inline:nn
        { one, two, three }
        { \item #1 }
\end{enumerate}
\ExplSyntaxOff
```

1. one
2. two
3. three

The expl3 syntax often seems cryptic to users of other programming languages. As a combination of underscores and colons and a bunch of naming conventions form a unique structure, expl3 is a nice way to remind everyone of the fact that LaTeX, and thus expl3, is a pure macro language. Here, tokens are replaced by their meaning and no actual operations are performed, as opposed to scripting languages.

Understanding the structure of expl3 therefore requires a basic understanding of macro expansion and the concept of category codes. The following sections explain the basics of these, in addition to the syntactic structure. If the concepts are already familiar, the corresponding sections can be skipped.

## 1 Syntax switching in TeX, LaTeX, expl3

When TeX processes input, it not only reads individual characters, but also assigns a category to each character. This category determines how the character should be processed. The assignment is done using the so-called “category codes” or “catcodes” for short. Each input character corresponds to a character code, and each character code is assigned to a (changeable) category code.

In total, TeX knows sixteen different categories. The assignments of a character to a category can change within a document. The most common example is language-dependent behavior, such as constructed by babel [1]. For German documents, one can type `"a` to produce “ä”. (In English documents, each character is processed separately.) This is done using category 13 “active”. Active characters are no longer simple characters, but commands; in this case, the command to put an umlaut over the following character.

The following list shows all available categories along with explanations and examples, many of which are familiar to all TeX typists, even if you haven’t heard of category codes.

0. Escape character (`\`)
1. Beginning of group (`{`)
2. End of group (`}`)
3. Math shift (`$`)
4. Alignment tab (`&`)
5. End of line (⟨return⟩)
6. Parameter (`#`)
7. Superscript (`^`)
8. Subscript (`_`)
9. Ignored character (⟨null⟩)
10. Space (␣)
11. Letter (non-ASCII only with XeTeX/LuaTeX)
12. Other character (`@`)
13. Active character (`~`)
14. Comment character (`%`)
15. Invalid character (⟨delete⟩)

doi.org/10.47397/tb/44-1/tb136peischl-expl3

### 1.1 The @ character in (LA)TEX macro names

LATEX uses @ to protect internal macros from access by end users. Usually, internal macros are protected this way for a reason. So customizations should be done with caution to avoid unexpected side effects. While being aware of danger, it is possible to use @ within command names, by changing the category code of @ to 11 (letter), and back to 12 (other) when no longer needed:

```
\makeatletter
\makeatother
```

A typical example for the use of the @ character is the definition of "starred" variants for one's own commands. For this, two auxiliary macros must be defined internally, which are often also protected with an @:

```
\makeatletter
\newcommand*{\cmd}
    {\@ifstar\@cmdstar\@cmd}
\newcommand*{\@cmd}{without *}
\newcommand*{\@cmdstar}{with *}
\makeatother
```

Then the macro `\cmd` has the following output:

```
\cmd \\ \cmd*
```

without *
with *

### 1.2 The expl3 syntax

Since the focus of the expl3 syntax is programming, LATEX behaves fundamentally differently when coding a command than when writing text:

- Spaces and newlines in code delimit tokens, but are otherwise ignored.
- Blank lines are not paragraph breaks.
- Tilde (~) characters are a normal space (catcode 10), not a tie.
- Colon (:) and underscore (_) characters are part of macro names.
- There is a syntactic difference between functions and variables.
- It is recommended to put spaces around curly braces unless they contain only one parameter.

These changes allow us to structure the code quite a bit better, without changing its meaning. Switching to (or from) the expl3 programming syntax is done with these commands:

```
\ExplSyntaxOn
\ExplSyntaxOff
```

Additionally, most package authors nowadays use CamelCase for the commands for their users. This improves the readability of the code even within the classic LATEX syntax.

## 2 Naming scheme

Using a naming scheme, expl3 distinguishes at a glance functions that process content or arguments from variables that simply store a value. Function names contain a : character, and variables do not:

**Functions:**

\*module_description*:*arguments*

**Variables:**

\*validity_module_description_datatype*

### 2.1 Variables

Variables store values. Expl3 provides different data types for this. The naming scheme is the same for all types. Technically, this is only a convention. However, following it ensures that users can understand the code more easily.

\*validity_module_description_datatype*

**Validity** constant (c), global (g) or local (l).

**Internal?** Internal variables which are not intended for end-users separate the *validity* from the *module* by two underscores. Normal variables have only one underscore here.

**Module** Named for the package/bundle to avoid name conflicts. There is a process to register module names; see [6].

**Description** What does the variable store for which purpose?

**Datatype** What kind of values are stored in the variable? How must it be processed?

An example of an internal variable (two underscores) is the token list (`tl`):

```
\l__siunitx_complex_sign_tl
```

There are a wide variety of data types, each with its own abbreviation. A small selection is shown in Table 1. All interfaces are documented in [4].

### 2.2 Functions

The term "function" is perhaps a bit misleading for those familiar with other programming languages. TEX and thus expl3 is a pure macro language, which

**Table 1**: Selection of expl3 data types

| Type | Description |
|---|---|
| `bool` | boolean (true or false) |
| `box` | box |
| `clist` | comma-separated list |
| `coffin` | "box with handles", a box with anchor points for relative placement with respect to other objects. |
| `dim` | length (dimension) |
| `fp` | floating point numbers (double precision) |
| `int` | integer |
| `prop` | key/value list (property list) |
| `seq` | sequence (queue/stack) |
| `skip` | extensible length (glue) |
| `str` | string, consisting only of letters, spaces and category 12 (other) characters |
| `stream` | input/output streams for reading/writing external files |
| `tl` | token list, string with arbitrary catcodes |

gives the impression of a function that operates only by substituting strings. However, functions in expl3 can also be understood to process their contents, with no return value in the sense of conventional programming languages. The "return value" is often left in the "input stream" and thus becomes part of the document.

The naming convention for functions is:

`\`*module_description*`:`*argument specification*

The *module* and *description* for function names are identical to those for variables. However, the difference between whether a function acts locally or globally is recorded in the description. The convention prefixes a `set` for a (local) assignment and an additional "`g`" for a global assignment: `gset`. Examples are the functions for setting integer variables:

```
\int_set:Nn
\int_gset:Nn
```

In expl3 syntax, the arguments expected by the function are specified after the colon. Thus, the two examples above expect two arguments: one of type "N" and a second of type "n". Thus, one can directly look at a function for the number and types of expected arguments. This will be important for controlling the expansion process, but for now, just N and n are enough to deal with. The letter stands for "No manipulation" in each case. Thus, the argument is passed to the function without any further processing.

The difference between the upper and lower case is whether the function expects a single token or a grouped argument. Upper case letters stand for tokens; lower case for groups. Thus, our argument specification `:Nn` above expects a single token followed by a group. Altogether, the macro `\int_set:Nn` could be used as follows:

```
\int_set:Nn \l_tmpa_int { 5 }
```

The above example sets an integer variable (the first argument, here `\l_tmpa_int`) to the value "5" (specified in a group).

The function is thus similar to classic LaTeX's `\setcounter`, but the argument of `\int_set:Nn` can also be used to calculate. The function `\int_use:N` returns the value of the given variable, in this case typesetting it:

```
\ExplSyntaxOn
\int_set:Nn \l_tmpa_int { 5 + 2 * 3 }
\int_use:N \l_tmpa_int
\ExplSyntaxOff
```

```
11
```

### 2.3 dim: Example of a data type

This section is devoted to the "dim" data type for lengths. The most common functions are discussed. Analogous functions exist for other data types. Complete documentation is given in [4].

#### 2.3.1 Initialization: N/n arguments

```
\dim_new:N
\dim_const:Nn
```

`\dim_new:N` is used to create a new variable. It then exists globally, but can also be assigned locally. Here, the naming convention is crucial. To create a local and a global variable for our present tutorial, we do:

```
\dim_new:N \l_tugboat_test_dim
\dim_new:N \g_tugboat_test_dim
```

It is thus possible for two variables with the same *description* to exist, one of which is to be assigned locally and one globally.

When defining a constant, its value is also directly specified:

```
\dim_const:Nn \c_tugboat_test_dim { 5cm }
```

For variables, which are changeable, the assignment is done separately with `\dim_set:Nn` (or `_gset`):

```
\dim_set:Nn \l_tugboat_test_dim { 3cm }
\dim_gset:Nn \g_tugboat_test_dim
                          { 1cm + 5mm }
```

Analogously to what we saw with counters, calculation is also possible here. Precision is limited to that of TEX for lengths. The smallest unit is 1 sp = 0.000 02 pt. And thus definitely small enough to have more than sufficient accuracy for print production.

Besides assignments, there are also commands for addition and subtraction of lengths. The name structure remains identical here. All details can be looked up in [4].

### 2.3.2 Conditionals and loops: T/F/TF arguments

Another basic part of programming is the ability to compare values of variables, and create loops based on such conditions. For lengths, the simplest way to do this in expl3 is to use the command:

```
\dim_compare:nTF
```

This macro can be used to compare lengths with each other. The possible comparisons are:

| | |
|---|---|
| Equal | `=` or `==` |
| Greater-equal | `>=` |
| Larger | `>` |
| Less-equal | `<=` |
| Smaller | `<` |
| Unequal | `!=` |

There are other variants of the `\dim_compare:` function that allow only some of the operators; they are processed faster. Here, we discuss only the simplest and most general command. As a complete example, our just-created and assigned global and local lengths can be compared with each other:

```
\ExplSyntaxOn
\dim_compare:nTF { \g_tugboat_test_dim >=
                   \l_tugboat_test_dim }
    { is~greater~or~equal }
    { is~smaller }
\ExplSyntaxOff
```

```
is smaller
```

The specification TF expects one group or token per letter according to the previous description. Here, the “T” stands for “true”, the “F” for “false”.

Expl3 has a special feature here: you are allowed to specify only the branch that is needed. If the function should output something only if the query returns the value “false”, the T argument can simply be omitted:

```
\ExplSyntaxOn
\dim_compare:nF
        { \g_tugboat_test_dim >
          \l_tugboat_test_dim }
    { not~greater }
\ExplSyntaxOff
```

```
not greater
```

### 2.3.3 Debugging outputs

In more complex programming, it may be necessary to display values of variables on the fly. Expl3 provides commands to output the current value of a variable in the terminal or to write it to the log file.

```
\dim_show:N
\dim_show:n
\dim_log:N
\dim_log:n
```

The variants with type “n” arguments expect a length expression instead of a length variable. Here you can calculate again or evaluate a macro which contains for example a centimeter value.

## 2.4 Argument specifications

In addition to the argument specifications already mentioned, there are several others, most of which process arguments differently than in standard arguments in classic LATEX. Table 2 shows all of them and their description.

`N` and `TF` have already been explained. Type `c` follows next, and section 4 will explain the specifications `o`, `f`, `x`, `e`, and `V`. The types `p` and `w` are less common, especially for beginners, and won’t be discussed here; they’re listed for the sake of completeness. Explanations can be found in [5].

**Table 2**: Argument specifications for expl3 functions

| Token | Description |
|---|---|
| `wN/n` | No manipulation |
| `TF/T/F` | True/False |
| `c` | Csname |
| `V/v` | Value |
| `o` | expand Once |
| `x` | eXhaustive expansion |
| `e` | Exhaustive expansion, but the macro might be expandable |
| `f` | Full expansion to first unexpandable token |
| `p` | Parameters as for TEX definitions |
| `w` | Weird |

## 3 Csname/Endcsname: `c` arguments

A fundamental concept of LaTeX is the ability for macro names to be created dynamically:

```
\csname name\endcsname
```

One can roughly describe the functionality of this construction as prefixing a backslash:

```
\LaTeX{}
=
\csname LaTeX\endcsname
```

```
LaTeX = LaTeX
```

A typical example of practical use is references. Here, a macro is defined internally that contains the argument of the `\label` command:

```
\label{frame:csname}
\expandafter\meaning
    \csname r@frame:csname\endcsname
```

```
macro:->{3}{91}{Csname\slash    Endcsname:
\texttt {c} arguments}{section.3}{}
```

A reference is thus created as a macro containing the element number — in this case empty because not numbered — and the page number. For `\ref`, the first value is used. `\pageref` uses the second. For more information on the topic, including several examples, see [3].

In expl3, this concept is the foundation for type `c` arguments — `c` as in "csname". Here, we've seen the first command, `\dim_set:Nn`, which sets `\l_tmpa_dim` to `1cm`. The second command, using `\dim_set:cn`, similarly sets `\l_tmpb_dim` to `2cm`, but the variable is given as a name instead of a literal control sequence. The third uses names for both the variable name and the value.

```
\dim_set:Nn \l_tmpa_dim { 1cm }
\dim_set:cn { l_tmpb_dim } { 2cm }
\dim_set_eq:cc { l_tmpa_dim }
               { l_tmpb_dim }
```

## 4 Controlling expansion: `o`/`x`/`e` arguments

Expansion essentially means replacing a command with its meaning. For classic commands in LaTeX, the expansion of a macro can be seen by the commands

```
\meaning\command
\show\command
```

The result can be displayed in the text or output to the terminal. For example:

```
\newcommand*{\myVariable}{myvalue}
\meaning\myVariable\\
\newcommand*{\myFunction}[1]{%
    function with argument (#1)
}
\meaning\myFunction
```

```
macro:->myvalue
macro:#1->function with argument (#1)
```

The macro `\myVariable` is thus simply replaced by the string "myvalue". The macro `\myFunction` accepts an argument and places this behind the text in parentheses.

To implement more complex structures, other macros are often used within macro definitions. Then, multi-level expansion is necessary.

### 4.1 Multiple-level expansion

We use the following definitions as an example to illustrate how LaTeX handles commands.

```
\newcommand\one{a}
\newcommand\two{\one,b}
\newcommand\three{\one,\two,c}
```

Imagine each macro as a box with different content depending on its definition:

Without expansion, TeX sees only a token. If this is then to be processed further, the box is unpacked:

Since there is only the word "one" inside this macro, this process cannot be repeated. The macro is already fully and exhaustively expanded.

Compared to the once-expandable macro `\one`, `\three` can be expanded multiple times. See Figure 1 for all steps.

Expl3 allows for arguments to explicitly control how far boxes should be unpacked before further processing. This allows for exact control.

Accordingly, we can now distinguish the argument specifications. We see this using an expl3 construct directly: `\tl_to_str:n` directly prints the argument, as a string, into the document. To get variants of the expansion levels, one can use `\exp_args:No` or other expansion levels. Here the first argument is not expanded and the second one is expanded according to the argument:

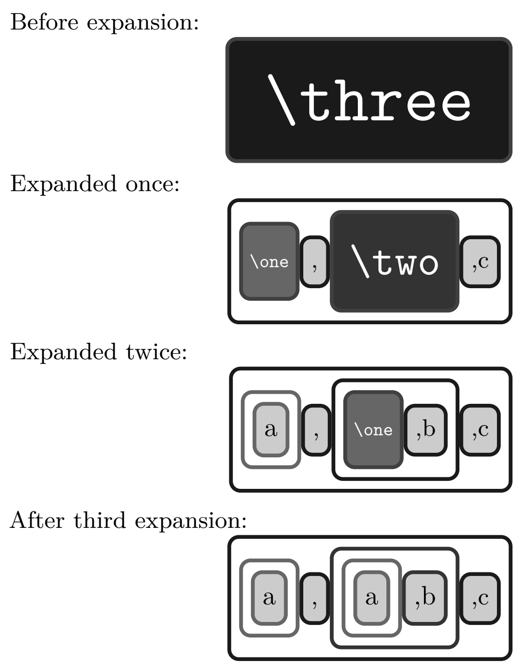


**Figure 1**: Visualization of expansion steps in boxes

```
\ExplSyntaxOn
unexpanded:\hfill
\tl_to_str:n { \three } \\
 expanded~once:\hfill
\exp_args:No \tl_to_str:n { \three } \\
fully~expanded:\hfill
\exp_args:Nf \tl_to_str:n { \three } \\
exhaustively~expanded~(x)\hfill
\exp_args:Nx \tl_to_str:n { \three } \\
exhaustively~expanded~(e)\hfill
\exp_args:Ne \tl_to_str:n { \three }
\ExplSyntaxOff
```

```
unexpanded:                       \three
expanded once:                \one ,\two ,c
fully expanded:                  a,\two ,c
exhaustively expanded (x)          a,a,b,c
exhaustively expanded (e)          a,a,b,c
```

This ability to control exactly in which order arguments of functions should be expanded makes it possible to look inside boxes before they are finally processed. Thus, for example, you can check in which format a date is passed and proceed accordingly:

```
\cs_new:Nn \__tugboattut_parse_date:n {
    % split at "-"
    \seq_set_split:Nnn \l_tmpa_seq
        { - } {#1}
    % more than one item
    % -> there was a dash
    \int_compare:nTF
        { \seq_count:N \l_tmpa_seq > 1 }
    {
        % assuming ISO format
        \seq_item:Nn \l_tmpa_seq { 3 } .
        \seq_item:Nn \l_tmpa_seq { 2 } .
        \seq_item:Nn \l_tmpa_seq { 1 }
    } {
        % alternative checks possible
        #1
    }
}
```

The macro now checks if the argument contains a hyphen. If it does, the date is interpreted as an ISO date (YYYY-MM-DD). Otherwise it is assumed that the date already has the format DD.MM.YYYY. (Further checking could be done.)

As an example, let's imagine the date that is passed at the beginning of the document via `\date` for the title line is to be processed with this. Internally, this value is stored in the command `\@date`. However, this macro does not contain a hyphen, which means that it has to be expanded first.

One way would be to use the `\exp_args:` command used above, but expl3 additionally provides a mechanism to create variants of a base command:

```
\cs_generate_variant:Nn
    \__tugboattut_parse_date:n
    { x }
```

Now the variant `\__tugboattut_parse_date:x` also exists. This allows the following construct:

```
\ExplSyntaxOn
\__tugboattut_parse_date:n { 23.06.2022 }
=
\__tugboattut_parse_date:n { 2022-06-23 }
=
\__tugboattut_parse_date:x
    { \use:c { @date } }
\ExplSyntaxOff
```

```
23.06.2022=23.06.2022=23.06.2022
```

## 5 Summary/outlook

LaTeX3 or expl3 extends and simplifies the ability to write more flexible macros despite — or maybe even because of — its somewhat weird syntax. In addition to the basic data type of "Token", structured data types are introduced, and a differentiation is made between data processing and data storage. Moreover, expl3 makes the control of macro expansion much more transparent.

It is thus possible to process different contents automatically and to pack significantly more functionality into a macro depending on the arguments and their structure. Furthermore, the uniform structure of the interfaces makes it easier to read and understand code by other authors.

Very many helper macros, which have been defined in many packages again and again, are obsolete, since there now are functions within the LaTeX kernel to replace them. Whole groups of packages are replaced by LaTeX3 standard modules, one of the most common examples being the key–value parsing packages [2]. In the long run, this — along with other extensions to the kernel — will reduce conflicts between packages and thus make the overall LaTeX system even more stable and increase the usability for the end user.

This article provides only a very small glimpse of the possibilities. Although more or less anything could be done in classic LaTeX, it often required low-level patches and hacking. Expl3 has made it much easier for me to do programming tasks directly in LaTeX.

I sincerely hope this article can help to reduce some confusion and fear, on the road to more expl3.

`\prg_do_nothing:` or just `\relax`.

## References

[1] J.L. Braams, J. Bezos. Babel: Localization and internationalization. `ctan.org/pkg/babel`

[2] CTAN. `keyval` topic. `ctan.org/topic/keyval`

[3] A. Hendrickson. The joy of `\csname... \endcsname`. *TUGboat* 33(2):219–224, 2012. `tug.org/TUGboat/tb33-2/tb104hendrickson.pdf`

[4] LaTeX Project. The LaTeX3 interfaces. `mirrors.ctan.org/macros/latex/contrib/l3kernel/interface3.pdf`

[5] LaTeX Project. The expl3 package and LaTeX3 programming. `mirrors.ctan.org/macros/latex/contrib/l3kernel/expl3.pdf`

[6] J. Wright. Registering expl3 module[s]. `texdev.net/2012/11/04/registering-expl3-module/`

◇ Marei Peischl
Gneisenaustr. 18
20253 Hamburg
Germany
`marei (at) peitex dot de`
`https://peitex.de`

# LuaCAS: Symbolic computation in LaTeX

Timothy All, Evan Cochrane

## Abstract

LuaCAS is a portable computer algebra system, written entirely in Lua, designed for use within LuaLaTeX via the `luacas` package [1]. Features include: arbitrary precision integer and rational arithmetic, number-theoretic algorithms, constructors and algorithms for univariate polynomials defined over various rings, symbolic differentiation and integration, and more.

## 1 Motivation

Most existing computer algebra systems such as Maple and Mathematica allow for converting their stored expressions to LaTeX code. But this still requires exporting code from LaTeX to another program, converting it to a form that the CAS is expecting, performing the computation, and importing the result, which can be tedious.

In contrast, the `luacas` package allows the user to perform basic symbolic computations from *within* LuaLaTeX without the need for laborious and technical setup. One simply installs the package like any other and adds `\usepackage{luacas}` to the preamble. Indeed, this article, along with all computations contained therein, was prepared in Overleaf.

## 2 An example

The main method for interacting with LuaCAS from within LuaLaTeX is to use the `CAS` environment. The following example demonstrates typical usage:

```
\begin{CAS}
  vars('x')
  f = int(sin(sqrt(x)),x)
\end{CAS}
\[ \print{f} = \print*{f} \]
```

$$\int \sin\left(\sqrt{x}\right) dx = -2\sqrt{x}\cos\left(\sqrt{x}\right) + 2\sin\left(\sqrt{x}\right)$$

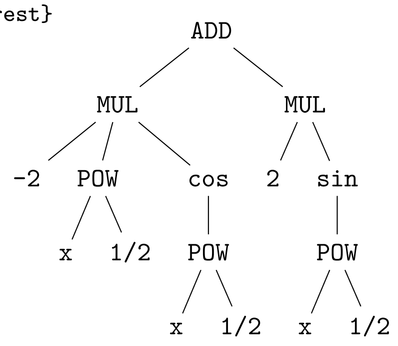
The macro `\print` converts the contents of its argument as-is into a string formatted for LaTeX and prints the result into the document via `tex.print`; the starred variant `\print*` evaluates and performs some basic simplifications on its argument before the conversion to LaTeX step.

## 3 Development

LuaCAS began as a senior capstone at Rose-Hulman Institute of Technology in the fall of 2021. The primary use case we had in mind early in development was that of a professor creating content with dynamic examples and problems for introductory calculus classes. We thus decided to make symbolic differentiation/integration the end goal of the project, as well as including basic algebraic functionality expected in any CAS.

Our first task was programming the core trees that would be used to store expressions. Mathematically speaking, an expression is a rooted tree. The `luacas` package comes with a pair of macros `\parseforest` (plus a starred variant that evaluates and simplifies the argument) and `@\forestresult` that allow the user to draw these rooted trees with the help of the `forest` package [9]. For example, with `f` defined as in the previous example, we have:

```
\parseforest*{f}
\bracketset{action character=@}
\begin{forest}
    for tree = {font=\ttfamily}
    @\forestresult
\end{forest}
```



Object-oriented Lua was chosen as our programming paradigm, allowing for the functionality of the CAS to be easily extensible without the need for compiling or setting up a complicated build environment.

Given that LuaCAS is written in an interpreted language, one can expect dramatically slower performance when comparing LuaCAS to popular computer algebra systems. Any mathematical operation or structure also gets its own class under this scheme, so there is an inevitable explosion in the number of classes. Despite this, LuaCAS performs well within the scope of its design and motivation.

We decided to split LuaCAS into modules, partly to increase the potential for extensibility and partly to reduce the time to load the CAS when only a subset of its features are needed. At the time of this writing, there are three modules: `core`, `algebra`, and `calculus`. The `luacas` package loads these modules by default.

The `core` module contains all of the interfaces for expressions that other classes need to extend (e.g., expression manipulation, substitution, etc.). The `algebra` module contains polynomial algorithms and common algebraic and trigonometric functions, as well as interfaces for algebraic structures such as

doi.org/10.47397/tb/44-1/tb136all-luacas

rings and fields. Class inheritance was chosen to mirror mathematical structures — fields inherit from Euclidean domains since all fields are Euclidean domains, and Euclidean domains likewise inherit from rings. Concrete classes represent specific rings, such as integers or polynomials, and objects are elements of these rings. Finally, the `calculus` module contains classes for differentiation and integration.

Features were implemented chronologically by an informal notion of how mathematically 'fundamental' they were. Accordingly, expression simplification via algebraic properties came first, then polynomial factoring/root-finding, then symbolic differentiation, and finally symbolic integration. This order turned out to be convenient for testing purposes, since symbolic integration relies on factoring for rational function integration. Algorithms for symbolic factoring and differentiation are well-established [10], but symbolic integration required significantly more tinkering to balance power and efficiency. This took the project beyond its original scope as a senior project; development on the symbolic integrator continued for months. Version 1.0.1 of `luacas` was uploaded to CTAN on November 15, 2022.

## 4 Features

The `CAS` environment is fundamentally a glorified `\directlua`. Accordingly, the `CAS` environment can be used essentially anywhere in a LaTeX document, and variables declared in one instance of the `CAS` environment will be remembered in the next instance of the `CAS` environment. Thus, expressions can be manipulated naturally throughout a document, as the examples below will illustrate.

### 4.1 Core

At the core of any computer algebra system is the notion of an *expression.* In LuaCAS, there are *atomic expressions* (e.g., integers, variables) and *composite expressions.* Variables must be declared or initialized before use (like Sage in days of yore). This is done with the `vars()` function within the `CAS` environment. Using a combination of Lua's meta-methods and operator overloading, composite expressions are constructed naturally:

```
\begin{CAS}
  vars('x')
  f,g = 1-x+0*x, 1+1*x
  h = f*g
\end{CAS}
\[ \print{h} \]
```

$$(1-x+0\cdot x)\,(1+1x)$$

Essential functionality for any computer algebra system is the process of *simplification.* Externally, the user expects output from the CAS to be simple and concise. Internally, simplification serves as a sort of normalization procedure for expressions. Expressions can be simplified in a couple of different ways:

```
\begin{CAS}
  ah = h:autosimplify()
  sh = h:simplify()
\end{CAS}
\[ \print{ah} = \print{sh} \]
```

$$(1+x)\,(1-x)=1-x^2$$

The `autosimplify` method is designed to be fast, as it is automatically performed on expressions before most other functions, such as factoring and expansion. Accordingly, it perhaps does not go as far as one might expect. For a more rigorous simplification, the `simplify` method makes a rudimentary search for the smallest expression tree equivalent to the input.

The core functionality of LuaCAS allows for other types of expression manipulation including substitutions:

```
\begin{CAS}
  vars('h')
  sh = substitute({[x]=x+h},sh)
\end{CAS}
\[ \print{sh} \]
```

$$1-(x+h)^2$$

### 4.2 Algebra

The Algebra module contains constructors for various special classes and related algorithms.

#### 4.2.1 Rings

There are constructors for the following Ring types:

- the integers,
- the integers modulo $N$,
- rationals (interpreted somewhat broadly), and
- polynomials.

A rudimentary parser wrapped around the contents of the `CAS` environment calls most of these constructors in a natural way:

```
\begin{CAS}
  a,b,c = 65, Mod(65,4), 63/65
\end{CAS}
\[ \lprint{{a,b,c}} \]
```

$$65, 1, \frac{63}{65}$$

But to construct a polynomial requires specific input from the user:

```
\begin{CAS}
  vars('x')
  f,g = x^2+2*x+3, Poly({3,2,1},x)
\end{CAS}
\[ \print{f} \qquad \print{g} \]
```

$$x^2+2x+3 \qquad x^2+2x+3$$

The printouts of `f` and `g` look the same; but internally, LuaCAS handles these expressions quite differently. There are several advantages to having a dedicated `polynomial` class, not least of which is computational speed.

On the other hand, the user more often than not needn't worry about issues pertaining to class types. Many functions in LuaCAS are class-aware in that they will either detect or make some attempt at converting class types for you. For example, the `factor` function applied to $a=1440$ will detect that the input is an `Integer`, then apply number theoretic algorithms to determine the prime-factorization (specifically a combination of Pollard-Rho and Miller-Rabin). On the other hand, when `factor` is given the expression $f=x^3+x^2+x-3$, it first converts $f$ to the `polynomial` class; from there it uses special algorithms to find the factorization over $\mathbf{Q}$ (specifically a combination of Berlekamp [2] and Zassenhaus [10]).

```
\begin{CAS}
  vars('x')
  a,f = 1440, x^3 + x^2 + x - 3
\end{CAS}
\[ \begin{aligned}
  \print{f} &= \print{factor(f)} \\
  \print{a} &= \print{factor(a)}
\end{aligned} \]
```

$$\begin{aligned} x^3+x^2+x-3 &= (-1+x)\left(3+2x+x^2\right) \\ 1440 &= 3^2 2^5 5^1 \end{aligned}$$

### 4.2.2 Ring conversion

Each Ring type comes equipped with a *Ring identifier.* This identifier is used to cast arithmetic performed on differing Ring types to the appropriate Ring. For example, if we ask LuaCAS to add a `polynomial` with integer coefficients to a `rational` number, LuaCAS will fetch the Ring identifiers for both classes and determine that the appropriate Ring into which to cast the arithmetic is the `polynomial` ring with `rational` coefficients:

```
\begin{CAS}
  a,b,c = 2, 4/3, Poly({-3,1,1,1},x)
  d = c+b+a
\end{CAS}
\[ (\print{c}) + \print{b}
  + \print{a} = \print{d}\]
```

$$(x^3+x^2+x-3)+\frac{4}{3}+2 = x^3+x^2+x+\frac{1}{3}$$

### 4.2.3 Special classes

The Algebra module provides constructors for special classes such as those for trigonometric, radical, and logarithmic expressions, along with support for the expected simplifications of these expressions:

```
\begin{CAS}
  vars('x')
  a,b,c=sin(4*pi/3),ln(e^(x+1)),sqrt(8/9)
\end{CAS}
\[ \begin{aligned}
  \print{a} &= \print*{a} \\
  \print{b} &= \print*{b} \\
  \print{c} &= \print*{c}
\end{aligned} \]
```

$$\begin{aligned} \sin\left(\frac{4\pi}{3}\right) &= -\frac{\sqrt{3}}{2} \\ \ln\left(e^{x+1}\right) &= 1+x \\ \sqrt{\frac{8}{9}} &= \frac{2\sqrt{2}}{3} \end{aligned}$$

### 4.2.4 Number theoretic algorithms

LuaCAS also provides basic number theoretic functionality. For example, LuaCAS can run the extended Euclidean algorithm:

```
\begin{CAS}
  a,b = 42250, 46137
  c,x,y = gcdext(a,b)
\end{CAS}
\[ \print{c} = \print{a} (\print{x})
    +\print{b}(\print{y}) \]
```

$$169 = 42250(-95)+46137(87)$$

LuaCAS also contains factoring algorithms and primality checking for the `Integer`-class. For primality checking, we use Miller-Rabin [5] and the base set of prime witnesses $p=2,3,\ldots,41$. Accordingly, primality checking can be trusted for integers a bit beyond $10^{24}$. For factoring, we use Miller-Rabin combined with Pollard-Rho [4] to search for prime factors recursively:

```
\begin{CAS}
  a = 407808999
  b = factor(a)
\end{CAS}
\[ \print{a} = \print{b} \]
```

$$407808999 = 3^4 31^3 13^2$$

### 4.2.5 Polynomial algorithms

LuaCAS hosts a number of algorithms for (univariate) polynomial arithmetic over the rationals or modulo a prime, including: extended Euclidean algorithm, factoring, and resultants.

```
\begin{CAS}
  vars('x')
  f = Mod(topoly(x^2+x+1),7)
  ff = factor(f)
\end{CAS}
```

```
\[ \print{f} = \print{ff} \]
```

$$x^2+x+1 = 1\left(x+5\right)^1\left(x+3\right)^1$$

LuaCAS also contains algorithms for symbolic root finding over the rationals (including supporting algorithms like those for finding decomposition series).

```
\begin{CAS}
  vars('x')
  f = topoly(x^4 + 4*x^3 - 8*x + 3)
  r = roots(f)
\end{CAS}
\[ \left\{ \lprint{r} \right\} \]
```

$$\left\{1, -3, -1+\sqrt{2}, -1-\sqrt{2}\right\}$$

### 4.3 Calculus

The Calculus module contains constructors for derivatives/integrals and algorithms for symbolic differentiation/integration.

#### 4.3.1 Differentiation

Due to the nature of differentiation, LuaCAS can quickly compute the derivatives of almost any expression that can be represented in LuaCAS.

```
\begin{CAS}
 vars('x', 'y', 'z')
 f,g = 3*ln(y)*sin(x), x^(1/(x*z))*x
 dg, df = diff(g,x), diff(f,{x,3},{y,2})
\end{CAS}
\[ \print{dg} = \print*{dg} \]
\[ \print{df} = \print*{df} \]
```

$$\frac{d}{dx}\left(x^{\frac{1}{xz}}x\right) = x^{1+\frac{\frac{1}{x}}{z}}\left(\frac{\left(1+\frac{\frac{1}{x}}{z}\right)}{x} - \frac{\ln(x)}{x^2z}\right)$$

$$\frac{\partial^5}{\partial y^2 \partial x^3}\left(3\ln(y)\sin(x)\right) = \frac{3\cos(x)}{y^2}$$

#### 4.3.2 Integration

LuaCAS can evaluate a wide variety of definite and indefinite integrals. The integrator mostly works by calling standard methods familiar to any college calculus student recursively (such as $u$-substitution, integration-by-parts, etc.) and then searching for the appropriate anti-derivative. For integration of rational functions, we use the method of Lazard, Rioboo, Rothstein and Trager [8].

```
\begin{CAS}
  f = e^(2*x)*cos(3*x)
  F = int(f, x,0,pi)
\end{CAS}
\[ \print{F} = \print*{F} \]
```

$$\int_0^{\pi} e^{2x}\cos(3x)\,dx = -\frac{2}{13} - \frac{2e^{2\pi}}{13}$$

However, we cannot guarantee that an integral will be able to be evaluated, even if the expression is integrable in elementary terms.

## 5 Interaction with the LaTeX ecosystem

Given that the CAS environment is just a glorified lua environment, LuaCAS interacts very well with TeX primitives and standard macros as well as the Lua language. Indeed, the design of LuaCAS was (in part) inspired by the potential to write reusable code such as:

```
\newcommand{\Euclid}[3]{%
\begin{CAS}
  vars('x')
  a,b,p = #1,#2,#3
  a = Mod(topoly(a),p)
  b = Mod(topoly(b),p)
  tex.print("\\[ \\begin{aligned}")
  while b.degree>0 do
    q,r = a:divremainder(b)
    tex.print(a:tolatex(),
      "&= (",
      b:tolatex(),
      ")(",
      q:tolatex(),
      ")+",
      r:tolatex(),
      "\\\\")
      a,b = b,r
    end
  tex.print("\\end{aligned} \\]")
\end{CAS}%
}
\Euclid{x^4+x^3+x^2+x+1}{x^3+2*x+3}{7}
```

$$\begin{aligned}
x^4-x^2+1 &= (x^3+2x+1)(x)+4x^2-x+1\\
x^3+2x+1 &= (4x^2-x+1)(2x+4)+4x+4\\
4x^2-x+1 &= (4x+4)(x+4)+6
\end{aligned}$$

The macro `\Euclid` displays the Euclidean algorithm applied to the polynomials found in the first and second arguments modulo the prime found in the third argument.

The `luacas` package comes with the macros `\fetch` and `\store`. These macros allow the user to pull content out of LuaCAS in a format that's appropriate for packages like TikZ/PGF [7] and Asymptote [3]. For example:

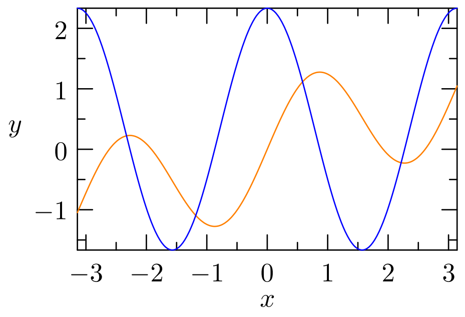
```
\begin{CAS}
  vars('x')
  f = sin(2*x)+x/3
  df = diff(f,x):autosimplify()
\end{CAS}
\store{f}\store{df}
```

The macro `\df` contains the string:

```
1/3 + (2 * (cos(2 * x)))
```

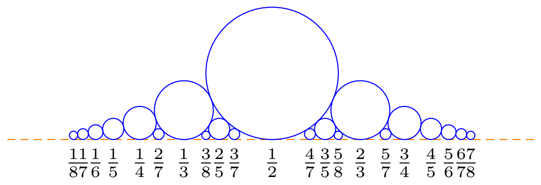
Macros created via the `\store` command can be called into other environments like the `asypicture` environment from the `asypictureB` package [6]:

```
\begin{asypicture}{}
  import graph; size(6cm,0);
  real f(real x){return @f;}
  real df(real x){return @df;}
  draw(graph(f,-pi,pi,operator..),orange);
  draw(graph(df,-pi,pi,operator..),blue);
  xaxis("$x$",BottomTop,LeftTicks);
  yaxis("$y$",LeftRight,RightTicks);
\end{asypicture}
```



The macro `\fetch` does nearly the same thing as `\store` except no macro is created; in other words, `\fetch{df}` can be used wherever we would have used `\df`. This is particularly useful for grabbing values out of tables built with LuaCAS:

```
\begin{tikzpicture}[scale=7]
  \draw[orange,densely dashed] (0,0) -- (1,0);
  \foreach \k in {2,...,22}{
    \draw[blue]
      (\fetch{F[\k]},\fetch{H[\k]})
      circle (\fetch{H[\k]});
    \node[below] at (\fetch{F[\k]},0)
      {\small$\print{F[\k]}$};
  }
\end{tikzpicture}
```


$\frac{1}{8}$ $\frac{1}{7}$ $\frac{1}{6}$ $\frac{1}{5}$ $\frac{1}{4}$ $\frac{2}{7}$ $\frac{1}{3}$ $\frac{3}{8}$ $\frac{2}{5}$ $\frac{3}{7}$ $\frac{1}{2}$ $\frac{4}{7}$ $\frac{3}{5}$ $\frac{5}{8}$ $\frac{2}{3}$ $\frac{5}{7}$ $\frac{3}{4}$ $\frac{4}{5}$ $\frac{5}{6}$ $\frac{6}{7}$ $\frac{7}{8}$

## 6 Future

In the future, we aim to expand the feature set of LuaCAS and include at least a decent chunk of the functionality common to popular existing computer algebra systems. This may include:

- summation and product expressions
- symbolic limits
- symbolic differential equation solving
- irreducible factorization of multivariate polynomials
- logic & set theory
- symbolic linear algebra
- numeric functionality

On the LaTeX side of things, it would be good to include some amount of externalization so that LuaCAS performs computations only when needed and not at every compile.

## 7 Acknowledgements

We'd like to thank the Mathematics Department at Rose-Hulman Institute of Technology for their support throughout this project. A special thanks goes to Joseph Eichholz for many helpful discussions and constructive feedback.

## References

[1] T. All, E. Cochrane. *The Luacas package.* `ctan.org/pkg/luacas`

[2] E.R. Berlekamp. Factoring polynomials over finite fields. *Bell System Technical Journal* 46(8):1853–1859, 1967.

[3] J. Bowman, A. Hammerlindl. *The Asymptote package.* `asymptote.sourceforge.net`

[4] J.M. Pollard. A Monte Carlo method for factorization. *Nordisk Tidskr. Informationsbehandling (BIT)* 15(3):331–334, 1975.

[5] M.O. Rabin. Probabilistic algorithm for testing primality. *J. Number Theory* 12(1):128–138, 1980.

[6] C. Staats III. *The AsypictureB package.* `ctan.org/pkg/asypictureb`

[7] T. Tantau, C. Feuersänger, et al. *The PGF package.* `ctan.org/pkg/pgf`

[8] B.M. Trager. Algebraic factoring and rational function integration. In *Proceedings of the third ACM symposium on Symbolic and algebraic computation*, pp. 219–226, 1976.

[9] S. Živanocić. *The Forest package.* `ctan.org/pkg/forest`

[10] H. Zassenhaus. On Hensel factorization, I. *J. Number Theory* 1(3):291–311, 1969.

◇ Timothy All
Department of Mathematics
Rose-Hulman Institute of Technology
Terre Haute, IN 47803 USA
`timothy.all (at) rose-hulman dot edu`

◇ Evan Cochrane
`cochraef (at) rose-hulman dot edu`

# Attributes in Markdown

Vít Novotný

## Abstract

Markup languages provide only a finite set of elements, whereas the wants of users are infinite. To bridge this gap, markup languages allow users to extend them with attributes.

In this article, we introduce attributes in the lightweight markup language of Markdown. We also show how writers can type them and how coders can style them using the Markdown package for TEX.

## Introduction

Markup languages provide only a finite set of elements to writers. This is especially true in lightweight markup languages such as AsciiDoc, Org-mode, and Markdown, which use ASCII punctuation marks and other non-letter symbols for tags. As a result, writers are often left unable to express their intent using the markup language.

In many markup languages, users can add new elements using syntax extensions. For example, in the Markdown package for TEX, writers can add tables using `pipeTables` and `tableCaptions` options:

```
\documentclass{article}
\usepackage[pipeTables, tableCaptions]
           {markdown}
\begin{document}
\begin{markdown}
 Right | Left | Center
------:|:-----|:------:
   12  |  12  |   12
  123  |  123 |  123
    1  |  1   |    1
: Example table
\end{markdown}
\end{document}
```

Possible output:

**Table:** Example table

| Right | Left | Center |
|---:|:---|:---:|
| 12 | 12 | 12 |
| 123 | 123 | 123 |
| 1 | 1 | 1 |

Since version 2.17.0 from October 2022, users can also write their own syntax extensions in Lua [1, Section 2.2]. However, writing syntax extension is a costly process that requires advanced coding skills.

Furthermore, in some markup languages, users can also mix different markup languages. For example, in the Markdown package for TEX, writers can easily mix Markdown with YAML metadata, TEX commands, and HTML tags:

```
\documentclass{article}
\usepackage[jekyllData, html,
            rawAttribute, texMathDollars]
           {markdown}
\begin{document}
\begin{markdown}
---
title: |
  Example Document in YAML, `\TeX`{=tex},
  <abbr>HTML</abbr>, and Markdown
author: Vít Novotný
date: 2023-03-24
---
# Introduction
Use YAML for metadata, *Markdown* for text,
`\TeX`{=tex} for $math$ and formatting, and
<abbr>HTML</abbr> for extended markup.
\end{markdown}
\end{document}
```

However, mixing different markup languages makes the text more difficult to read, typeset, and less suitable for multitarget publishing.

Lastly, in most markup languages, users can attach attributes to elements to denote additional information. For example, in version 2.22.0 of the Markdown package for TEX from March 2023, writers can easily attach attributes to Markdown headings, text spans and blocks, inline code spans and code blocks, links, and images:

```
\documentclass{article}
\usepackage[
  bracketedSpans, fencedCode, fencedDivs,
  fencedCodeAttributes, headerAttributes,
  inlineCodeAttributes, linkAttributes
]{markdown}
\begin{document}
\begin{markdown}
Introduction {#introduction}
============
Here is an [important]{color=red} text
that describes the implementation of the
[Quicksort][1] algorithm in Haskell:

~~~~ haskell {.numberLines startFrom=100}
qsort []     = []
qsort (x:xs) = qsort (filter (< x) xs)
            ++ [x]
            ++ qsort (filter (>= x) xs)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


[1]: https://wikipedia.org/wiki/Quicksort
     {.external-link .wikipedia}
\end{markdown}
\end{document}
```

In the above example, attributes are written between pairs of curly braces ({}). When used in moderation, attributes can work around the shortcomings of markup languages without decreasing readability.

doi.org/10.47397/tb/44-1/tb136novotny-markdown-attr

In this article, we introduce attributes in Markdown. In Section 1, we describe the kinds of attributes available in Markdown and how writers can use them. In Section 2, we show how coders can style attributes using the Markdown package for TEX. In Section 3, we discuss the changes to attributes in the next major version of the Markdown package. We conclude in Section 4 by summarizing the contributions of the article.

## 1 Writer's workshop

In Markdown, writers can use three different kinds of attributes: identifiers, class names, and key-values.

Identifiers are the most common type of attributes. Writers can use identifiers to assign a unique label to an element and refer to it, similar to the `\label` and `\ref` commands in LATEX:

```
\documentclass{article}
\usepackage[headerAttributes,
            relativeReferences]{markdown}
\begin{document}
\begin{markdown}
We conclude in Section <#conclusion>.

Conclusion {#conclusion}
==========
In this paper, we have discovered that most
grandmas would rather eat dinner with their
grandchildren than get eaten. Begone, wolf!
\end{markdown}
\end{document}
```

Possible output:

We conclude in Section X.

**X Conclusion**

In this paper, we have discovered that most grandmas would rather eat dinner with their grandchildren than get eaten. Begone, wolf!

Unlike identifiers, class names do not uniquely identify an element. Instead, they place an element into a category. For example, writers can use a class name such as `fruit` to mark all occurrences of fruit:

```
\documentclass{article}
\usepackage[bracketedSpans]{markdown}
\begin{document}
\begin{markdown}
[Mango]{.fruit} is the king of all fruits.
[Oranges]{.fruit} are full of Vitamin C.
An [apple]{.fruit} a day keeps doctor away.
\end{markdown}
\end{document}
```

Even if we are undecided how the output should look, adding the attributes allows us to easily style all occurrences of fruit in our document later on.

Whereas class names denote coarse-grained category membership, key-values describe fine-grained traits of an element. For example, writers can use key-values such as `width` and `height` for image size:

```
\documentclass{article}
\usepackage[linkAttributes]{markdown}
\begin{document}
\begin{markdown}

![](example-image){width=3cm height=2cm}

\end{markdown}
\end{document}
```

## 2 Coder's cubicle

In version 2.22.0 of the Markdown package, writers can attach three different types of attributes to seven different types of elements. To prevent a combinatorial explosion, attributes and element types are encoded separately in the abstract syntax tree of a document. Consider the following example document:

```
\documentclass{article}
\usepackage[bracketedSpans, linkAttributes,
            inlineCodeAttributes]{markdown}
\begin{document}
\begin{markdown}

[This text]{.red} is so important it glows
red (grayscaled for print). So does this
<https://link>{.red} and this `code`{.red}.

\end{markdown}
\end{document}
```

The document would produce the following abstract syntax tree:

```
\bracketedSpanAttributeContextBegin
  \attributeClassName{red}%
  This text%
\bracketedSpanAttributeContextEnd{}
is so important it glows red
(grayscaled for print). So is this
\linkAttributeContextBegin
  \attributeClassName{red}%
  \link{https://link}%
       {https://link}%
       {https://link}%
       {}%
\linkAttributeContextEnd{}
and this
\codeSpanAttributeContextBegin
  \attributeClassName{red}%
  \codeSpan{code}%
\codeSpanAttributeContextEnd
```

This allows us to easily style the class name red independently on the element that it is attached to:

```
\ExplSyntaxOn
\markdownSetup {
  renderers = {
    *ContextBegin = {
      \color_group_begin:
    },
    attributeClassName = {
      \str_if_eq:nnT
        { #1 } { red }
        { \color_select:n { red } }
    },
    *ContextEnd = {
      \color_group_end:
    },
  }
}
\ExplSyntaxOff
```

Output:

This text is so important it glows red (grayscaled for print). So does this `https://link` and this `code`.

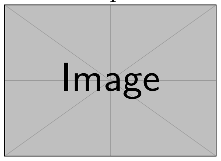
By contrast, consider the last document from the previous section, which would produce the following abstract syntax tree:

```
\imageAttributeContextBegin
  \attributeKeyValue{height}{2cm}%
  \attributeKeyValue{width}{3cm}%
  \image{}{example-image}{example-image}{}%
\imageAttributeContextEnd
```

Here, we want to style the key-values width and height only for images:

```
\RequirePackage{graphicx}
\ExplSyntaxOn
\markdownSetup {
  renderers = {
    imageAttributeContextEnd = {
      \group_end:
    },
    imageAttributeContextBegin = {
      \group_begin:
      \markdownSetup {
        renderers = {
          attributeKeyValue = {
            \setkeys  % Pass the key-value
              { Gin } % to graphicx package
              { { ##1 } = { ##2 } }
          },
        },
      }
    },
  },
}
\ExplSyntaxOff
```

Output:



## 3 Developer's den

For all Markdown elements except headings, the *AttributeContextBegin and End renderers immediately surround the element in the abstract syntax tree. By contrast, the headerAttributeContextEnd renderer is placed after the end of the section implied by the heading. While this is practical for styling whole sections [2, Section 2.4], it is inconsistent and makes other common use cases such as expanding the attributeIdentifier renderer to the \label LaTeX command more difficult to implement.

In version 2.21.0 of the Markdown package from February 2023, the sectionBegin and End renderers were added, which surround sections implied by headings regardless of whether attributes are used. In version 3.0.0 of the Markdown package, currently scheduled for release around June 2023, the headerAttributeContextEnd renderer will appear immediately after headings in abstract syntax tree.

## 4 Conclusion

Attributes provide a simple bottom-up mechanism for extending markup languages with new concepts. In this article, we have shown the types of attributes that are available in the lightweight markup language of Markdown. We have also shown how writers can type attributes in their documents and how coders can style attributes using the Markdown package for TeX. With attributes, writers can produce beautiful documents without littering them with formatting commands.

### References

[1] V. Novotný. Markdown 2.17.1: What's new, what's next? *TUGboat* 43(3):276–278, 2022. doi.org/10.47397/tb/43-3/tb135novotny-markdown

[2] V. Novotný, D. Rehák, et al. Markdown 2.15.0: What's new? *TUGboat* 43(1):10–15, 2022. doi.org/10.47397/tb/43-1/tb133novotny-markdown

◇ Vít Novotný
Studená 453/15
Brno, 638 00
Czech Republic
witiko (at) mail dot muni dot cz
github.com/witiko

# An introduction to automata design with Ti*k*Z's automata library

Igor Borja

## Abstract

This article is a quick introduction to Ti*k*Z's *automata* library, used for the design and typesetting of finite automata in LaTeX. It also explores the use of TeX loops and conditionals to automate the generation of images that follow noticeable patterns. Ti*k*Z itself is a package used for generating a variety of figures — from geometry configurations to graphs and automata — allowing for more control over image editing and quality. Although the package is very versatile, its uses for designing automata will be the primary topic of this article.

## 1 Introduction and basic syntax

Finite automata, also called finite state machines, are a basic concept in computer science for modeling computation. Wikipedia (`en.wikipedia.org/wiki/Finite-state_machine`) provides an introduction to the topic.

In this article, all the code to typeset an automaton will be contained inside a `tikzpicture` environment [3]. After starting the environment, you can pass optional arguments, separated by a comma, such as *node distance* and *arrow style*. A reminder: the node distance (which we'll see below) must be a dimension (`cm`, `em`, `pt`, etc.).

### 1.1 Nodes

You can declare a node of an automaton via the following syntax:

```
\node[state, ⟨state modifiers⟩,
      ⟨position modifiers⟩] (⟨id⟩) {⟨name⟩};
```

Note that a node declaration *should always end in a semicolon*. Let's analyse all of these parameters:

1. Every state node must begin with the word `state`.
2. State modifiers are mostly used to indicate that node is the `initial` node or an `accepting` node, and thus are often not needed.
3. Position modifiers are used to place a node relative to another (*already declared*) node. Some common modifiers are `right=of` ⟨*id*⟩, `left=of` ⟨*id*⟩, `below=of` ⟨*id*⟩ and `above=of` ⟨*id*⟩. Here, ⟨*id*⟩ is the id of the node relative to which the positioning is carried out.

   Also, it's possible to give dimensions via the `xshift` and `yshift` parameters to achieve manual control over the position after the relative placement.
4. The ⟨*id*⟩ is the *unique* identifier that will be used to refer to that node later.
5. The ⟨*name*⟩ is the text that will appear in the automaton, inside the circle that represents that node. It does not need to be plain text; it's also possible to use a math expression (enclosed by single `$` signs).

A minimal working example:

```
\documentclass{standalone}
\usepackage{tikz}
\usetikzlibrary{automata}
\usetikzlibrary{positioning, arrows}
\begin{tikzpicture}
    [->, node distance = 2cm]
    \node[state] (p) {$p = 1$};
    \node[state, accepting, right=of p,
        yshift= -2cm] (q) {$q = 2$};
\end{tikzpicture}
```

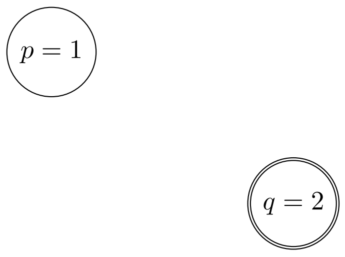


**Figure 1**: Two nodes

The use of ⟨*direction*⟩ `of=`⟨*id*⟩, although correct, is marked as deprecated in PGF/Ti*k*Z source code [4].

### 1.2 Edges

You can declare an edge between two nodes with the following syntax

```
    (⟨id-head⟩) edge[⟨options⟩]
    node[⟨options⟩]{⟨value⟩} (⟨id-tail⟩);
```

Every sequence of consecutive edge declarations must be preceded by a `\draw` command. Also, a semicolon is used to indicate the last edge declaration of a sequence; any that come after that and before another `\draw` command will be ignored. Therefore (provided `n1, n2, n3` have all been declared), this is correct:

```
\draw
(n1) edge node{text} (n2)
(n2) edge node{more text} (n3)
(n2) edge node{more text} (n1);
\node[state, above=of n1] (n4) {text};
\draw
(n3) edge node{more text} (n4);
```

While this is (for both reasons mentioned above) not correct:

```
\draw
(n1) edge node{text} (n2)
(n2) edge node{more text} (n3)
(n2) edge node{more text} (n1)
\node[state, above=of n1] (n4) {text};
(n3) edge node{more text} (n4);
```

The edges are treated as directed — to get the visual effect of undirected edges, remove the arrow in the environment options.

### 1.2.1 Analysing the different components of an edge command

⟨*id-head*⟩ is the identifier of the *head* node — the node from which the edge is originated.

⟨*id-tail*⟩ is the identifier of the *tail* node — the node to which the edge arrives.

The options after the `edge` keyword indicate how the edge should be drawn. The following options are the most common:

- Directions `right`, `left`, `below`, `above`: indicate from where the edge should leave
- `loop`: specifies that the edge should loop and go back to the head node. Using the `loop` option makes Ti*k*Z ignore the tail node's id (which can be left empty).

  Combining the `loop` option with directions indicates where the loop should be rendered: above, to the right, left or below the node.
- `bend`: indicates bends in the edge to a certain direction

The options after the `node` keyword indicate how the text associated with that edge should be drawn. The most common arguments are the four main directions `right`, `left`, `above` and `below`.

Defaults: if no `edge`-positioning options are given, the edge will be drawn as a straight line by default. Also, if no `node` options (i.e., options that determine the positioning of the edge text) are provided, the text will be rendered at the "center" of the edge by default. See both behaviors below:

```
\documentclass{standalone}
\usepackage{tikz}
\usetikzlibrary{automata}
\usetikzlibrary{positioning, arrows}
\begin{tikzpicture}
    [->, node distance = 2cm]
    \node[state] (p) {$p = 1$};
    \node[state, accepting, right=of p,
        yshift=-2cm] (q) {$q = 2$};
    \draw
    (p) edge node{hello} (q)
    (p) edge[loop] ();
\end{tikzpicture}
```

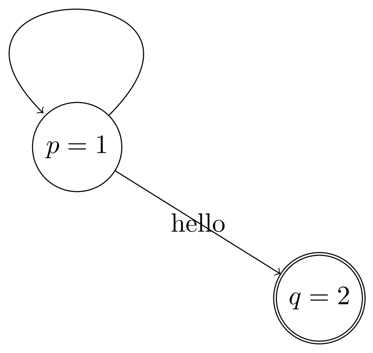


**Figure 2**: Two nodes, a labeled edge and a loop

## 2 Loops and automation by example

Here we show a more complex example.

### 2.1 Context

Consider the language $L$ over the alphabet $\Sigma = \{a, b\}$ that contains all the sequences with at least 2 characters $a$ and at least 1 character $b$. Note that $L$ is the intersection of two regular languages (over the same alphabet $\{a, b\}$): the set of strings with at least 2 characters $a$ and the set of strings with at least 1 character $b$.

Therefore, using the construction detailed in [2], a possible finite automaton that recognizes $L$ is $M = (Q, \{a, b\}, \delta, q_{0,0}, q_{2,1})$, where

$$Q = \left\{ q_{i,j} \,\middle|\, 0 \le i \le 2 \land 0 \le j \le 1 \right\}$$

are the states that can be reached. The transition function $\delta$ works as follows: $\delta(q_{i,j}, a) = q_{\min(2,i+1),j}$ and $\delta(q_{i,j}, b) = q_{i,\min(1,j+1)}$.

In other words, $q_{i,j}$ represents that the string read has (up to that point) $i$ characters $a$ (if $i < 2$) or 2 or more, if $i = 2$. Also, it has $j$ characters $b$ if $j < 1$, else it has 1 or more. Representing it graphically, we get the following state diagram:

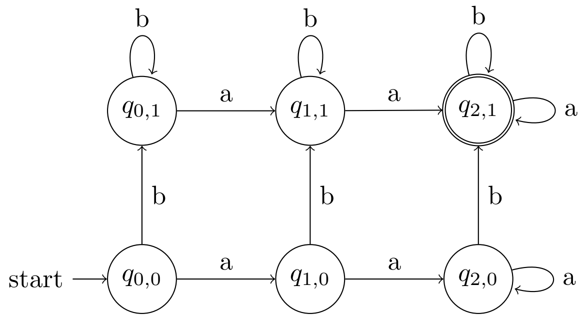


**Figure 3**: State diagram

which we can produce with the following LaTeX code:

```
\begin{tikzpicture}
    [->, node distance = 1.3cm]
    \node[state, initial] (a0b0)
        {$q_{0,0}$};
    \node[state, right=of a0b0] (a1b0)
        {$q_{1,0}$};
    \node[state, right=of a1b0] (a2b0)
        {$q_{2,0}$};
    \node[state, above=of a0b0] (a0b1)
        {$q_{0,1}$};
    \node[state, above=of a1b0] (a1b1)
        {$q_{1,1}$};
    \node[state, accepting, above=of a2b0]
        (a2b1) {$q_{2,1}$};
    %% Horizontal edges in first layer
    \draw
    (a0b0) edge node[above]{a} (a1b0)
    (a1b0) edge node[above]{a} (a2b0)
    (a2b0) edge[loop right]
        node[right]{a} (a2b0);
    %% First set of vertical edges
    \draw
    (a0b0) edge node[right]{b} (a0b1)
    (a1b0) edge node[right]{b} (a1b1)
    (a2b0) edge node[right]{b} (a2b1);
    %% Horizontal edges in second layer
    \draw
    (a0b1) edge node[above]{a} (a1b1)
    (a1b1) edge node[above]{a} (a2b1)
    (a2b1) edge[loop right]
        node[right]{a} (a2b1);
    %% Second set of vertical edges
    \draw
    (a0b1) edge[loop above]
        node[above]{b} (a0b1)
    (a1b1) edge[loop above]
        node[above]{b} (a1b1)
    (a2b1) edge[loop above]
        node[above]{b} (a2b1);
\end{tikzpicture}
```

It is easy to see that this state diagram's grid-like structure generalizes nicely to the family of languages $(L_{m,n})_{m,n\in\mathbb{N}}$, where $L_{m,n}$ is the language of strings with at least $m$ characters $a$ and at least $n$ characters $b$.

However, drawing the state diagram for $L_{m,n}$ quickly becomes too much work, since there will be $(m+1)(n+1)$ nodes and $2(m+1)(n+1)$ edges: even at small values (say, $m = 4$ and $n = 3$) it is still more than 60 lines of code.

## 2.2 Loops

However, due to its very simple structure, rendering automata like this one can be abstracted and automated in a relatively straightforward way, using `foreach` loops, available through the package `pgffor`.

1. For the nodes, it's possible to draw first $q_{0,0}$ with id `a0-b0`, then for each $1 \le i \le m$ draw $q_{i,0}$ at the right of $q_{i-1,0}$ and attribute to it the id `a`$\langle i\rangle$`-0` (where $\langle i\rangle$ is a placeholder for the value of $i$). This completes the first row.

   Then, for each $1 \le j \le n$ and for each $0 \le i \le m$ we draw $q_{i,j}$ above $q_{i,j-1}$ and attribute to it the id `a`$\langle i\rangle$`-b`$\langle j\rangle$. At each iteration it is necessary to check if $i = m$ and $j = n$, in which case we need to add the `accepting` option.

2. For the edges, we have four different cases: standard horizontal edges, standard vertical edges, edges that loop above the node and edges that loop at the right of the node.

   For each $0 \le i \le m-1$ and $0 \le j \le n$, we build a horizontal edge from $q_{i,j}$ to $q_{i+1,j}$. Then, for each $0 \le i \le m$ and $0 \le j \le n-1$ we build a vertical edge from $q_{i,j}$ to $q_{i,j+1}$. Finally, we build an edge that loops above $q_{i,n}$ for each $0 \le i \le m$ and an edge that loops at the right of $q_{m,j}$ for each $0 \le j \le n$.

It is necessary to refer to $i-1$ and $j-1$ several times in this algorithm. However, simply using the TeX code `\i - 1` and `\j - 1` and `a\i - 1b\j - 1` (for ids) in the TeX code won't work: loop indices in `foreach` loops function as standard variables defined with `\def` (just with restrained scope). That means that any reference `\i` will be replaced by the value of `\i` (as a string), so the expression `\i - 1` will not be evaluated. For example, if $i = 5$, `\i - 1` will be replaced by `5 - 1`.

In order to fix that, we use the fixed-point arithmetic package called `fp` and its command for evaluating expressions: `\fpeval`. The implementation below summarizes all that in working LaTeX code, abstracting it all in a command with three arguments called `\gridAutomata`. The first argument is the node distance, the second is $m$ and the third is $n$:

```
\usepackage{ifthen}
\newcommand{\gridAutomata}[3][2cm]
{
\begin{tikzpicture} [
    ->,
    node distance = #1,
]
%% building first row of nodes
\ifthenelse{\equal{#2}{0}}
{
    % if m == 0
    \ifthenelse{\equal{#3}{0}}
    {
        \node[state, initial,
            initial where=below, accepting]
            (a0-b0) {$q_{0,0}$};
    }{
        \node[state, initial,
            initial where=below]
            (a0-b0) {$q_{0,0}$};
    }
}{
    % if m > 0
    \node[state, initial, initial where=below]
        (a0-b0) {$q_{0,0}$};


    \foreach \i in {1,...,\fpeval{#2 - 1}}
    {
        \node[state,
            right=of a\fpeval{\i - 1}-b0]
            (a\i-b0) {$q_{\i, 0}$};
    }
    \ifthenelse{\equal{#3}{0}}
    {
    % if m > 0 and n == 0
        \node[state,
            right=of a\fpeval{#2 - 1}-b0,
            accepting]
            (a#2-b0) {$q_{#2, 0}$};
    }{
        % if m > 0, n > 0
        \node[state,
            right=of a\fpeval{#2 - 1}-b0]
            (a#2-b0) {$q_{#2, 0}$};
    }
}
%% other rows
\foreach \j in {1,...,#3}
{
    \foreach \i in {0,...,#2}
    {
        \ifthenelse{\equal{\i}{#2}
                    \AND \equal{\j}{#3}}
        {
            \node[state,
                above=of a\i-b\fpeval{\j-1},
                accepting]
                (a\i-b\j) {$q_{\i, \j}$};
        }{
            \node[state,
                above=of a\i-b\fpeval{\j-1}]
                (a\i-b\j) {$q_{\i, \j}$};
        }
    }
}
%% Constructing the edges
%% Loops above
\foreach \i in {0,...,#2}
{
    \draw (a\i-b#3) edge[loop above]
        node[above]{b} (a\i-b#3);
}
%% Rightmost loops
\foreach \j in {0,...,#3}
{
    \draw (a#2-b\j) edge[loop right]
        node[right]{a} (a#2-b\j);
}
%% Horizontal edges
\foreach \i in {0,...,\fpeval{#2 - 1}}
{
    \foreach \j in {0,...,#3}
    {
        \draw (a\i-b\j) edge
        node[above]{a} (a\fpeval{\i+1}-b\j);
    }
}
%% Vertical edges
\foreach \j in {0,...,\fpeval{#3 - 1}}
{
    \foreach \i in {0,...,#2}
    {
        \draw (a\i-b\j) edge
        node[right]{b} (a\i-b\fpeval{\j + 1});
    }
}
\end{tikzpicture}
}
```

Using this command for $m = 4, n = 3$ with a node distance of 1.5 cm yields the result below:

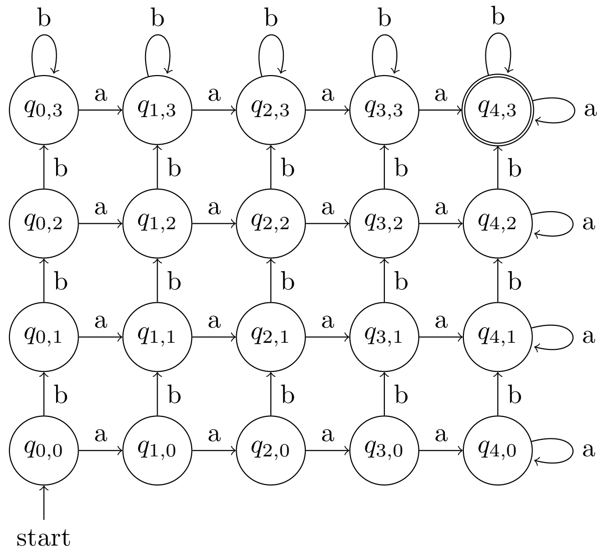


**Figure 4**: Result for $m = 4,\ n = 3$

This command's generality allows for construction of a larger example. The image on the next page uses the values $m = 7, n = 8$ with a node distance set to 1.5 cm.

$$\begin{tikzcd}
q_{0,8} \arrow[r, "a"] \arrow[loop above, "b"] & q_{1,8} \arrow[r, "a"] \arrow[loop above, "b"] & q_{2,8} \arrow[r, "a"] \arrow[loop above, "b"] & q_{3,8} \arrow[r, "a"] \arrow[loop above, "b"] & q_{4,8} \arrow[r, "a"] \arrow[loop above, "b"] & q_{5,8} \arrow[r, "a"] \arrow[loop above, "b"] & q_{6,8} \arrow[r, "a"] \arrow[loop above, "b"] & q_{7,8} \arrow[loop above, "b"] \arrow[loop right, "a"] \\
q_{0,7} \arrow[r, "a"] \arrow[u, "b"'] & q_{1,7} \arrow[r, "a"] \arrow[u, "b"'] & q_{2,7} \arrow[r, "a"] \arrow[u, "b"'] & q_{3,7} \arrow[r, "a"] \arrow[u, "b"'] & q_{4,7} \arrow[r, "a"] \arrow[u, "b"'] & q_{5,7} \arrow[r, "a"] \arrow[u, "b"'] & q_{6,7} \arrow[r, "a"] \arrow[u, "b"'] & q_{7,7} \arrow[u, "b"'] \arrow[loop right, "a"] \\
q_{0,6} \arrow[r, "a"] \arrow[u, "b"'] & q_{1,6} \arrow[r, "a"] \arrow[u, "b"'] & q_{2,6} \arrow[r, "a"] \arrow[u, "b"'] & q_{3,6} \arrow[r, "a"] \arrow[u, "b"'] & q_{4,6} \arrow[r, "a"] \arrow[u, "b"'] & q_{5,6} \arrow[r, "a"] \arrow[u, "b"'] & q_{6,6} \arrow[r, "a"] \arrow[u, "b"'] & q_{7,6} \arrow[u, "b"'] \arrow[loop right, "a"] \\
q_{0,5} \arrow[r, "a"] \arrow[u, "b"'] & q_{1,5} \arrow[r, "a"] \arrow[u, "b"'] & q_{2,5} \arrow[r, "a"] \arrow[u, "b"'] & q_{3,5} \arrow[r, "a"] \arrow[u, "b"'] & q_{4,5} \arrow[r, "a"] \arrow[u, "b"'] & q_{5,5} \arrow[r, "a"] \arrow[u, "b"'] & q_{6,5} \arrow[r, "a"] \arrow[u, "b"'] & q_{7,5} \arrow[u, "b"'] \arrow[loop right, "a"] \\
q_{0,4} \arrow[r, "a"] \arrow[u, "b"'] & q_{1,4} \arrow[r, "a"] \arrow[u, "b"'] & q_{2,4} \arrow[r, "a"] \arrow[u, "b"'] & q_{3,4} \arrow[r, "a"] \arrow[u, "b"'] & q_{4,4} \arrow[r, "a"] \arrow[u, "b"'] & q_{5,4} \arrow[r, "a"] \arrow[u, "b"'] & q_{6,4} \arrow[r, "a"] \arrow[u, "b"'] & q_{7,4} \arrow[u, "b"'] \arrow[loop right, "a"] \\
q_{0,3} \arrow[r, "a"] \arrow[u, "b"'] & q_{1,3} \arrow[r, "a"] \arrow[u, "b"'] & q_{2,3} \arrow[r, "a"] \arrow[u, "b"'] & q_{3,3} \arrow[r, "a"] \arrow[u, "b"'] & q_{4,3} \arrow[r, "a"] \arrow[u, "b"'] & q_{5,3} \arrow[r, "a"] \arrow[u, "b"'] & q_{6,3} \arrow[r, "a"] \arrow[u, "b"'] & q_{7,3} \arrow[u, "b"'] \arrow[loop right, "a"] \\
q_{0,2} \arrow[r, "a"] \arrow[u, "b"'] & q_{1,2} \arrow[r, "a"] \arrow[u, "b"'] & q_{2,2} \arrow[r, "a"] \arrow[u, "b"'] & q_{3,2} \arrow[r, "a"] \arrow[u, "b"'] & q_{4,2} \arrow[r, "a"] \arrow[u, "b"'] & q_{5,2} \arrow[r, "a"] \arrow[u, "b"'] & q_{6,2} \arrow[r, "a"] \arrow[u, "b"'] & q_{7,2} \arrow[u, "b"'] \arrow[loop right, "a"] \\
q_{0,1} \arrow[r, "a"] \arrow[u, "b"'] & q_{1,1} \arrow[r, "a"] \arrow[u, "b"'] & q_{2,1} \arrow[r, "a"] \arrow[u, "b"'] & q_{3,1} \arrow[r, "a"] \arrow[u, "b"'] & q_{4,1} \arrow[r, "a"] \arrow[u, "b"'] & q_{5,1} \arrow[r, "a"] \arrow[u, "b"'] & q_{6,1} \arrow[r, "a"] \arrow[u, "b"'] & q_{7,1} \arrow[u, "b"'] \arrow[loop right, "a"] \\
q_{0,0} \arrow[r, "a"] \arrow[u, "b"'] & q_{1,0} \arrow[r, "a"] \arrow[u, "b"'] & q_{2,0} \arrow[r, "a"] \arrow[u, "b"'] & q_{3,0} \arrow[r, "a"] \arrow[u, "b"'] & q_{4,0} \arrow[r, "a"] \arrow[u, "b"'] & q_{5,0} \arrow[r, "a"] \arrow[u, "b"'] & q_{6,0} \arrow[r, "a"] \arrow[u, "b"'] & q_{7,0} \arrow[u, "b"'] \arrow[loop right, "a"] \\
\text{start} \arrow[u] & & & & & & &
\end{tikzcd}$$

**Figure 5**: Result for $m = 7, n = 8$

## 3 More customization possibilities

### 3.1 Absolute positioning

It is also possible to define the position of each node manually, as a coordinate pair. The syntax is:

`\node[state] at (⟨x⟩, ⟨y⟩) (⟨id⟩) {⟨name⟩};`

The fields ⟨$x$⟩ and ⟨$y$⟩ are the $x$ and $y$ components of the position. To better illustrate this, we make use of the `help lines` option to draw a $3 \times 3$ grid, in which the nodes will be positioned.

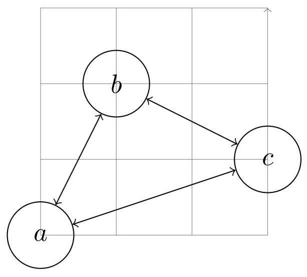


**Figure 6**: An isosceles triangle

```
\begin{tikzpicture}[<->]
    \draw[help lines] (0,0) grid (3,3);
    \node [state] at (0, 0) (a) {$a$};
    \node [state] at (1, 2) (b) {$b$};
    \node [state] at (3, 1) (c) {$c$};
    \draw
    (a) edge (b)
    (b) edge (c)
    (c) edge (a);
\end{tikzpicture}
```

### 3.2 Drawing arcs and bending edges

Especially when avoiding crossing edges in more complicated automata, it is useful to be able to draw edges as arcs (instead of straight lines). In plain Ti*k*Z it is possible to render arbitrary arcs, specifying the center, the starting and stopping angle, and the radius (as shown in [5]):

`\draw (⟨x⟩,⟨y⟩) arc (⟨start⟩:⟨stop⟩:⟨radius⟩);`

Through the automata library, however, the syntax is simplified for the case of an arc between two (already defined) nodes.

`(⟨id1⟩) edge[bend ⟨options⟩] (⟨id2⟩);`

More commonly, a directional option (`left` or `right`) is used to inform in which side the arc should be rendered. Furthermore, its radius can also be changed, by including the angle (in degrees) of the arc. Our last example illustrates these options:

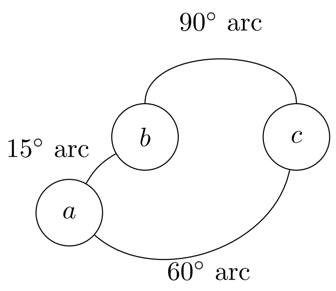


**Figure 7**: Various arcs

```
\begin{tikzpicture}
\node[state] at (0, 0) (a) {$a$};
\node[state] at (1, 1) (b) {$b$};
\node[state] at (3, 1) (c) {$c$};
\draw
(a) edge[bend left=15] node[above left]
{$15^{\circ}$ arc} (b)
(b) edge[bend left=90] node[above=0.25]
{$90^{\circ}$ arc}(c)
(c) edge[bend left=60] node[below]
{$60^{\circ}$ arc} (a);
\end{tikzpicture}
```

### References

[1] S. Sikdar. *Drawing Finite State Machines in LaTeX using TikZ: A Tutorial*, 2017.

[2] M. Sipser. *Introduction to the Theory of Computation*. Thomsom/Course Technology, 2005. Pages 48–50.

[3] T. Tantau, et al. *The TikZ and PGF Packages: Manual for version 3.1.10*, ch. Automata Drawing Library, pp. 571–575. 2023. `ctan.org/pkg/pgf`

[4] TeX Stack Exchange. Difference between "right of=" and "right=of" in PGF/TikZ. `tex.stackexchange.com/questions/9386`.

[5] TeX Stack Exchange. How is arc defined in TikZ? `tex.stackexchange.com/questions/175016`.

[6] TikZBlog. Automata diagrams in LaTeX. `latexdraw.com/automata-diagrams-in-latex`, 2021.

◇ Igor Borja
`igorpradoborja (at) gmail dot com`
`https://github.com/IgorPBorja`

# Styling R ggplot2 graphics with LaTeX

Travis Stenborg

## Abstract

The ggplot2 package is widely used for R graphics. Example LaTeX-style rendering of such graphics is presented, achieved via annotations with embedded LaTeX markup. This allows R graphics to be integrated into LaTeX documents with a harmonious visual style.

## 1 Technology stack

R is a statistical programming language. The material presented here was implemented using R 4.2.3, using the RStudio integrated development environment, on Windows 11.

In the same way that LaTeX is enhanced by supporting packages, R is also. Key R packages for integrating LaTeX into R graphics were extrafont, fontcm, ggplot2 and latex2exp.

The Ghostscript interpreter was used to embed default LaTeX fonts into PDF files. R, RStudio and Ghostscript are all free, and enjoy multiplatform support. Examples of these technologies applied to other TeX-related issues appear elsewhere in *TUGboat* [1, 4, 5].

## 2 Ghostscript setup

Ghostscript typically needs manual configuration on Windows, summarized here. First, ensure it's in your operating system's path, e.g.:

```
C:\Program Files\gs\gs10.00.0\bin
```

Also add the environment variables GS_CMD, identifying the Ghostscript executable, e.g.:

```
C:\Program Files\gs\gs10.00.0\bin\
    gswin64c.exe
```

and GS_FONTPATH, designating any font folders to use, e.g.:

```
C:\Windows\Fonts\
C:\Users\Travis\AppData\Local\Microsoft\
   Windows\Fonts\
```

An R application-level environment variable for Ghostscript's executable should be set too. Find your local Rprofile document (usually somewhere under the R installation directory) and append the new setting, e.g.:

```
Sys.setenv(R_GSCMD = "C:/Program Files/gs/
    gs10.00.0/bin/gswin64c.exe")
```

## 3 Computer Modern fonts in R

The ggplot2 package is designed especially for plots in R. A ggplot2 object is instantiated, associated with data, and its display properties specified programmatically.

Font properties of ggplot2 objects can be set in R to emulate LaTeX. A Computer Modern default font is assumed to be present [2].

```
ggplot2::theme(
    text = ggplot2::element_text(
        family = "CM Roman", size = 10
    )
)
```

To style such plots with Computer Modern fonts, a one-time R call installs them:

```
extrafont::font_install("fontcm")
```

To load the fonts in subsequent R sessions, make a relevant extrafont call at least once per session:

```
extrafont::loadfonts(quiet = TRUE)
```

## 4 Styling ggplot2

LaTeX-styled strings can be emulated in plots via the latex2exp package. Only a LaTeX subset is supported, enumerable in R.

```
latex2exp::latex2exp_supported()
```

The package accommodates common features such as math mode (with escaped backslashes), or inline Unicode.

```
latex2exp::TeX("weight \\textbf{W}_1$")
latex2exp::TeX("high\U00ADtech")
```

## 5 PDF rendering

PDF rendering is invoked via grDevices (loaded by default in R). Fonts were embedded in the output file via extrafont and Ghostscript. Finally, superfluous bounding whitespace can be cropped via knitr.

Example R code to style ggplot2 with LaTeX, showing the results of Markov Chain Monte Carlo convergence testing for a mixture model, as per e.g., [3], is given below. The resulting PDF appears in Figure 1.

```
# Data setup.
library(latex2exp)
labels <- c(
 TeX("$\\eta_2$"), TeX("$\\eta_1$"),
 TeX("$\\sigma_2$"), TeX("$\\sigma_1$"),
 TeX("$\\mu_2$"), TeX("$\\mu_1$"))
df <- data.frame(
 x = c(3040, 3040, 3458,
       3392, 2758, 4176),
 y = sapply(labels, deparse))

# Fonts and PDF driver setup.
extrafont::loadfonts(quiet = TRUE)
file_name <- "plot_example.pdf"
pdf(file_name)

# Build plot.
library(ggplot2)
grid_line <- element_line(
 linewidth = 0.25, linetype = "dashed",
 color = "grey")
plot <- ggplot(df, aes(x = x, y = y)) +
 geom_point() +
 scale_y_discrete(labels = labels) +
 theme_bw() + theme(
  axis.text = element_text(
   color = "black"),
  panel.grid.major = grid_line,
  panel.grid.minor = grid_line,
  text = element_text(
   family = "CM Roman", size = 10)) +
 xlab(TeX(
  "bulk\U00AD{}$\\textit{n}_{eff}$")) +
 ylab("Mixture parameter")

# Set plot size.
gridExtra::grid.arrange(
 grobs = lapply(list(plot),
  egg::set_panel_size,
  width = grid::unit(42, "mm"),
  height = grid::unit(42, "mm")))

# Close extraneous graphics devices.
while (!is.null(dev.list())) {
 device_num <- as.integer(dev.cur())
 if (device_num != 1) {
  dev.off(which = device_num)}}

# Finalise plot.
extrafont::embed_fonts(file_name)
knitr::plot_crop(file_name)
```

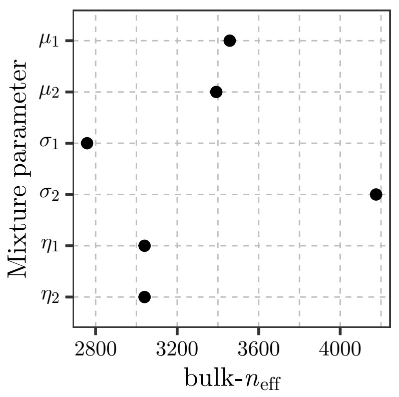


**Figure 1**: Example R ggplot2 output. The y-axis tick mark labels and x-axis label were styled using LaTeX.

## Acknowledgements

This work was supported by the Australian Research Council Training Centre in Data Analytics for Resources and Environments (project ICI9010031) and funded by the Australian Government (including contributions by the Australian National Health and Medical Research Council Ideas Grant GNT1186572).

## References

[1] L. Scarso. Two applications of SWIGLIB: GraphicsMagick and Ghostscript. *TUGboat* 36(3):237–242, 2015. tug.org/TUGboat/tb36-3/tb114scarso.pdf

[2] W. Schmidt. Font selection in LaTeX: The most frequently asked questions. *TUGboat* 28(2):241–242, 2007. tug.org/TUGboat/tb28-2/tb89schmidt.pdf

[3] A. Vehtari, A. Gelman, et al. Rank-normalization, folding, and localization: An improved $\widehat{R}$ for assessing convergence of mcmc (with discussion). *Bayesian Analysis* 16(2):667–718, June 2021. doi.org/10.1214/20-BA1221

[4] B. Veytsman. Using knitr and LaTeX for literate laboratory notes. *TUGboat* 43(2):130–133, 2022. tug.org/TUGboat/tb43-2/tb134veytsman-labnotes.pdf

[5] U. Ziegenhagen. Dynamic reporting with R/Sweave and LaTeX. *TUGboat* 31(2):189–192, 2010. tug.org/TUGboat/tb31-2/tb98ziegenhagen.pdf

◇ Travis Stenborg
Sydney, Australia
ORCID 0000-0002-2693-9628

# Creating annotated Unicode-like font charts

Janusz S. Bień

## Abstract

Printing annotated font tables in the layout following the Unicode standard documentation is discussed. Existing tools are presented and desirable but missing features are proposed.

## 1 Unicode charts

In this article, we will discuss the primary Unicode charts;[1] the secondary charts describing the variation sequences, in particular those belonging to Ideographic Variation Database, are outside the scope of our interest.

All the charts are produced with the Unibook formatting software,[2] supplied by ASMUS, Inc. (a company owned by Asmus Freytag, Technical Vice President of the Unicode Consortium from 1991 to 2007). A version of Unibook is available free of charge (but with a rather complicated license); it is used to prepare proposals of new characters, as well as the full standard. The program runs on several versions of MS Windows. The fonts used to print the characters themselves in the charts come from many sources and are not, in general, free in any sense.[3]

The charts proper are followed by various annotations. The source of annotations, at least in principle, is the `NamesList.txt`[4] file belonging to the Unicode Character Database. Its header states: *This file is semi-automatically derived from UnicodeData.txt and a set of manually created annotations using a script to select or suppress information from the data file.* The documentation of the format of the file is provided.[5] However, there are systematic discrepancies between the content of the file and the charts, so somewhere in the workflow additional processing is performed.

As of the standard version 15.0, there are the following annotations used (examples are given later):

- Lines starting with • (U+2022 BULLET) are just comment lines.
- Lines starting with = (U+003D EQUALS SIGN) are alias lines.
- Lines starting with ※ (U+203B REFERENCE MARK) are formal alias lines.[6]
- Lines starting with → (U+2192 RIGHTWARDS ARROW) contain cross-references to related characters.
- Lines starting with ≡ (U+2261 IDENTICAL TO) are used for precomposed characters and contain the list of characters which together compose the character in question.[7]
- Lines starting with ≈ (U+2248 ALMOST EQUAL TO) are used for so-called compatibility characters[8] and show the compatibility decomposition of the character (this relation is defined by enumeration).
- Lines starting with ~ (U+007E TILDE) describe registered variation sequences.

The file also contain commands to control printing titles, subtitles, block headers, various subheaders and notices.

## 2 The fntsample tool and its extension

In 2007 Eugeniy Meshcheryakov (the original spelling seems to be Евгений Мещеряков) released the first version of the fntsample program.[9] The tool was developed for the DejaVu Fonts project.

The charts it generates resemble those of the Unicode standard; see Fig. 1. In particular, they include glyphs for the characters which are invisible by definition, such as VS1 (U+FE00 VARIATION SELECTOR-1), if the font contains them.

In 2013 a student of mine, Paweł Parafiński, accepted the task to made the samples even more similar to the Unicode charts by supplementing them with annotations. He solved the problem in two steps. First he created a parser for the `NamesList.txt` file which converted it into simple XML. In the second step he extended fntsample to print the additional information from the XML file.

The program is orphaned, but the repositories are still available;[10] in particular, this allows reporting the issues. The most annoying problem is the inability to break long character names. There was also a problem with the parser, which was coded in the now-obsolete Python 2. I managed to adapt it to Python 3 (by trial and error, as I am not a programmer).

[1] `unicode.org/charts`
[2] `unicode.org/unibook`
[3] `unicode.org/charts/fonts.html`
[4] `unicode.org/Public/UNIDATA/NamesList.txt`
[5] `unicode.org/Public/UCD/latest/ucd/NamesList.html`
[6] Formal aliases are used for control characters, which have no glyphs and official names. Another use is correcting name mistakes; because of the stability principle, erroneous names are not removed from the standard, but the correct names are added as a formal alias.
[7] Some characters 'decompose' into another single character, but this is another story; the most well-known example is U+212B ANGSTROM SIGN, which 'decomposes' into U+00C5 LATIN CAPITAL LETTER A WITH RING ABOVE.
[8] See `en.wikipedia.org/wiki/Unicode_compatibility_characters`, for example.
[9] `github.com/eugmes/fntsample`
[10] Now at `github.com/jsbien/fntsample-with-comments` and `github.com/jsbien/ucd_xml_parser`.

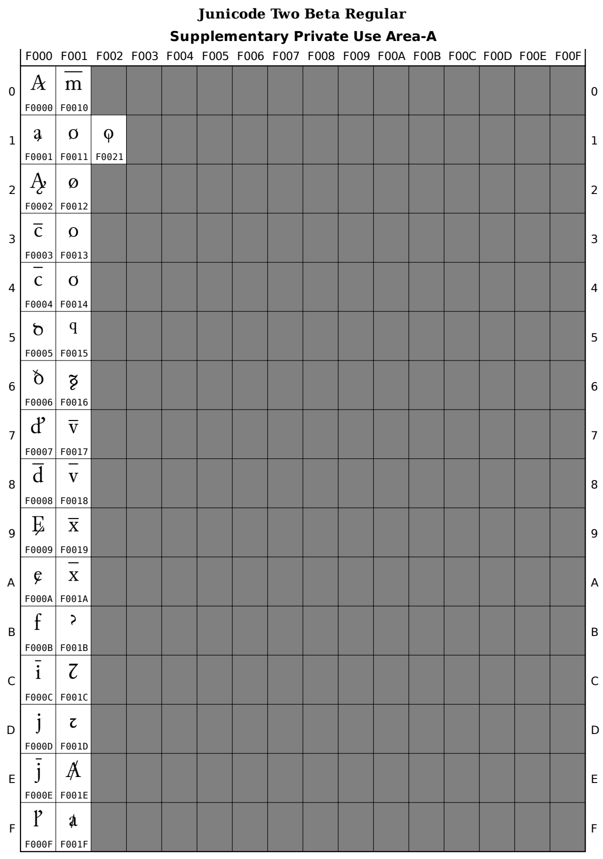

Junicode Two Beta Regular
Supplementary Private Use Area-A

**Figure 1**: Output of
`fntsample -f JunicodeTwoBeta-Regular.ttf -i 0xf0000- -o sample.pdf`

Unfortunately the parser ignores some elements of the `NamesList.txt` file, as at the time of its design they seemed not to be needed.

## 3 Frank Mittelbach's unicodefonttable

Frank Mittelbach's unicodefonttable package is available from CTAN and the usual distributions; its source repository is `github.com/frankmittelbach/fmitex-unicodefonttable`. It provides a LaTeX style file to include a font table in a document and, an interactive standalone version to produce a font table as a separate document. Tables can be supplemented by block names and user captions; tables can be limited to selected blocks.

The unicodefonttable package is implemented using the fontspec package, which is an advantage, as this allows using all font features which the package supports. The source of the Junicode font manual [1], typeset with XƎLaTeX, gives extensive examples of the usage of font features with the fontspec package.[11] They are also discussed in the manual [1, p. 13]:

[11] `github.com/psb1558/Junicode-font`

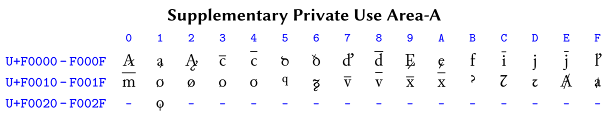

Supplementary Private Use Area-A

**Figure 2**: unicodefonttable output for plane 15 in `JunicodeTwoBeta-Regular.ttf`

> Many OpenType features produce different outcomes depending on an index passed to an application's layout engine along with the feature tag. Different applications have different ways of entering this index: consult your application's documentation. Here, the index is recorded in brackets after the feature tag. Users of fontspec (with XƎLaTeX or LuaTeX) should also be aware that fontspec indexes start at zero while OpenType indexes start at one. Therefore all index numbers listed in this document must be reduced by one for use with fontspec.

For referencing a specific value of a feature we follow [1], e.g., cv02[1] means the feature cv02 (character variant number 2) with the index 1; on the other hand ss10 is an example of a feature (stylistic set number 10) which does not require an index but which is just on or off.

It is worth noting that fontspec, and hence also unicodefonttable, if used with LuaLaTeX, allow for the little known "raw feature" `-invisible`,[12] allowing printing glyphs for characters which are invisible in principle (mentioned earlier; see also, for example, [3]) if the font contains them.[13]

Figure 2 shows the output of the following:

```
\displayfonttable
  [range-start=F0000,range-end=FFFFF,
   nostatistics,noheader,hex-digits=block]
  {JunicodeTwoBeta-Regular.ttf}
```

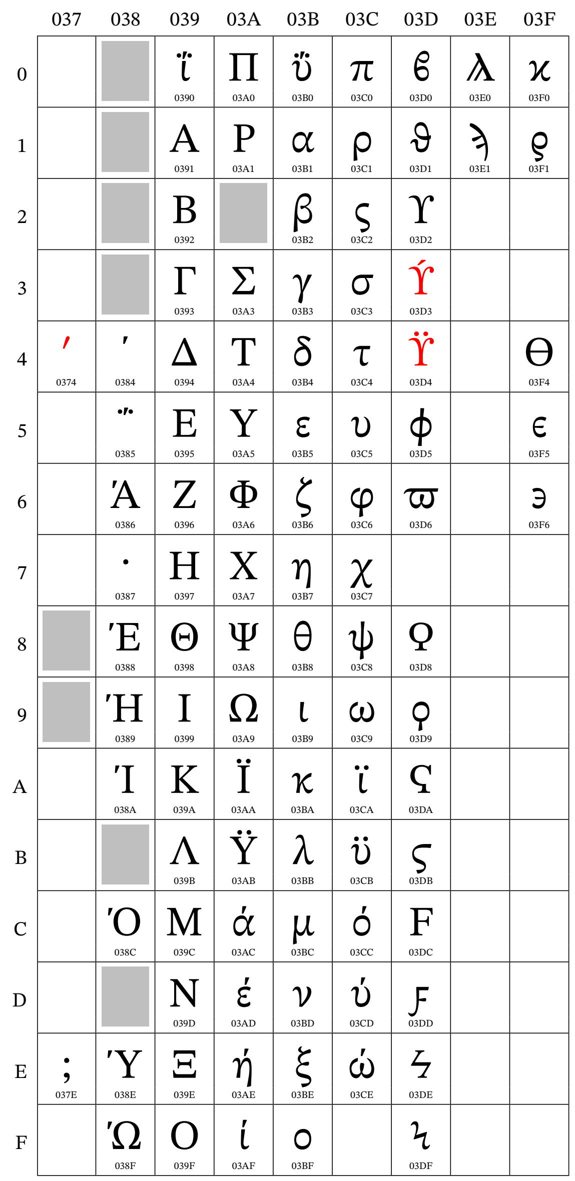
## 4 David M. Jones' STIX charts

The charts for the STIX fonts (*OpenType Unicode fonts for Scientific, Technical, and Mathematical texts*[14]) are typeset by David M. Jones with XƎLaTeX from source generated by a Perl script. The principal part of the charts has the same layout as the Unicode charts, so this technique could be used to replace fntsample (the additional part of the charts describes the OpenType features, which is only indirectly related to the present paper). Figure 3 shows the page for the Greek and Coptic block, starting at U+037.

[12] `github.com/latex3/luaotfload/issues/63`

[13] I learned this from the author; see `github.com/FrankMittelbach/fmitex-unicodefonttable/issues/5`.

[14] `github.com/stipub` and `www.stixfonts.org`



| | 037 | 038 | 039 | 03A | 03B | 03C | 03D | 03E | 03F |
|---|---|---|---|---|---|---|---|---|---|
| 0 | | ▨ | ΐ 0390 | Π 03A0 | ΰ 03B0 | π 03C0 | ϐ 03D0 | Ϡ 03E0 | ϰ 03F0 |
| 1 | | ▨ | Α 0391 | Ρ 03A1 | α 03B1 | ρ 03C1 | ϑ 03D1 | ϡ 03E1 | ϱ 03F1 |
| 2 | | ▨ | Β 0392 | ▨ | β 03B2 | ς 03C2 | ϒ 03D2 | | |
| 3 | | ▨ | Γ 0393 | Σ 03A3 | γ 03B3 | σ 03C3 | ϓ 03D3 | | |
| 4 | ʹ 0374 | ΄ 0384 | Δ 0394 | Τ 03A4 | δ 03B4 | τ 03C4 | ϔ 03D4 | | ϴ 03F4 |
| 5 | | ΅ 0385 | Ε 0395 | Υ 03A5 | ε 03B5 | υ 03C5 | ϕ 03D5 | | ϵ 03F5 |
| 6 | | Ά 0386 | Ζ 0396 | Φ 03A6 | ζ 03B6 | φ 03C6 | ϖ 03D6 | | ϶ 03F6 |
| 7 | | · 0387 | Η 0397 | Χ 03A7 | η 03B7 | χ 03C7 | | | |
| 8 | ▨ | Έ 0388 | Θ 0398 | Ψ 03A8 | θ 03B8 | ψ 03C8 | Ϙ 03D8 | | |
| 9 | ▨ | Ή 0389 | Ι 0399 | Ω 03A9 | ι 03B9 | ω 03C9 | ϙ 03D9 | | |
| A | | Ί 038A | Κ 039A | Ϊ 03AA | κ 03BA | ϊ 03CA | Ϛ 03DA | | |
| B | | ▨ | Λ 039B | Ϋ 03AB | λ 03BB | ϋ 03CB | ϛ 03DB | | |
| C | | Ό 038C | Μ 039C | ά 03AC | μ 03BC | ό 03CC | Ϝ 03DC | | |
| D | | ▨ | Ν 039D | έ 03AD | ν 03BD | ύ 03CD | ϝ 03DD | | |
| E | ; 037E | Ύ 038E | Ξ 039E | ή 03AE | ξ 03BE | ώ 03CE | Ϟ 03DE | | |
| F | | Ώ 038F | Ο 039F | ί 03AF | ο 03BF | | ϟ 03DF | | |

**Figure 3**: A page from `STIXTwoText-Regular.pdf` (headers and footers are omitted).

These charts use some additional conventions, e.g., red (grayscaled for print) is used for characters not directly supported but synthesized by a Unicode-aware shaping engine; this is shown with characters U+03D3 and U+03D4 in the figure; U+0374 is another example of a character which "decomposes" into (is substituted by) another single character.[15]

At present neither the XƎLATEX source nor the Perl script are available publicly.[16]

[15] `github.com/stipub/stixfonts/issues/248`
[16] `github.com/stipub/stixfonts/issues/247`

## 5 Typesetting character annotations

Although in the Unicode standard the character annotations occur immediately after the relevant glyph tables, it is not necessary to follow the standard in this respect. Separately provided annotations can be even more convenient.

Typesetting individual annotations in a way approximating the Unicode charts is not difficult. After some research I decided to use xltabular, which handles multipage tables (unfortunately it doesn't work in a multicolumn environment). The examples in figures 4–6 are typeset using code like this (showing just the first line; formatting has been slightly changed):

```
\begin{xltabular}{1.0\linewidth}%
                 {lll>{\raggedright}X}
0000 & &\multicolumn{2}{l}{<control>}\\
&    & &*& NULL\\
```

The Unicode conventions can be used easily to describe Private Use Area characters, e.g., those from Medieval Unicode Font Initiative[17] (other interesting initiatives are CONSCRIPT[18] and LINCUA[19]), or from a specific font. Fig. 5 shows some examples.

The line format for the registered variation sequences can be easily adapted to tag variation sequences (see, for example, [2]) and font specific information, as shown in Fig. 6.

The information about the font can be skipped when not needed.

## 6 Providing character annotations

In addition to the ttx program discussed later, the other tool I know of to list font features in a human-readable form is layout-features.py,[20] but its simple output does not contain the information needed for our purposes (see Fig. 7).

In 2018 a feature request was submitted to the fntsample repository entitled *Suggestion: enable printing glyphs not assigned to a unicode slot*,[21] but there was no follow up. In 2021 a similar feature request was made for unicodefonttable.[22] The answer, in my view correct, was that this should be a separate project.

We focus here on discussing the annotations in ttx's XML form (but the fork of layout-features.py may

[17] `mufi.info`
[18] `www.kreativekorp.com/ucsur`
[19] `bit.ly/2XVTzRL-LINCUA`
[20] `github.com/fonttools/fonttools/blob/8ab6af03c89726cf80ca3c4b755ae1bd0038b5da/Snippets/layout-features.py`; see also `github.com/fonttools/fonttools/discussions/2873`.
[21] `github.com/eugmes/fntsample/issues/11`
[22] `github.com/FrankMittelbach/fmitex-unicodefonttable/issues/2`

| | | | |
|---|---|---|---|
| 0000 | | <control> | |
| | | ※ | NULL |
| 0030 | 0 | DIGIT ZERO | |
| | | ~ | 0030 FE00 0︀ short diagonal stroke form |
| 0040 | @ | COMMERCIAL AT | |
| | | = | at sign |
| 0104 | Ą | LATIN CAPITAL LETTER A WITH OGONEK | |
| | | ≡ | 0041 A 0328 ◌̨ |
| 2105 | ℅ | CARE OF | |
| | | ≈ | 0063 c 002F / 006F o |
| 2B7A | ⭺ | LEFTWARDS TRIANGLE-HEADED ARROW WITH DOUBLE HORIZONTAL STROKE | |
| | | ※ | LEFTWARDS TRIANGLE-HEADED ARROW WITH DOUBLE VERTICAL STROKE |
| A7C1 | ꟁ | LATIN SMALL LETTER OLD POLISH O | |
| | | • | used in Old Polish as a nasal vowel |
| | | → | 00F8 ø LATIN SMALL LETTER O WITH STROKE |

**Figure 4**: Typesetting standard character annotations.

| | | | |
|---|---|---|---|
| F1C8 |  | COMBINING ABBREVIATION MARK ZIGZAG ABOVE CURLY FORM | |
| | | • | MUFI since v.2 |
| E8AF |  | LATIN SMALL LIGATURE LONG S L WITH STROKE | |
| | | • | MUFI in v.4 at E8DF, later moved to E8AF |
| | | • | used in old Polish |
| F0001 | 󰀁 | LATIN SMALL LETTER A WITH STROKE THROUGH TERMINAL | |
| | | • | supported in Junicode Beta since Jun. 30, 2020 |
| | | • | used in old Polish |
| | | → | 0105 ą LATIN SMALL LETTER A WITH OGONEK |
| | | → | 2C65 ⱥ LATIN SMALL LETTER A WITH STROKE |

**Figure 5**: Some examples from Unicode's Private Use Area.

| | | | |
|---|---|---|---|
| 0062 | b | LATIN SMALL LETTER B | |
| | | ~ | supported in Junicode Beta since Aug. 25, 2022 with ss10 on: 0062 b E0070 󠁰 E0073 󠁳 b old Polish *b quadratum* |
| 0105 | ą | LATIN SMALL LETTER A WITH OGONEK | |
| | | • | Polish, Lithuanian, ... |
| | | ≡ | 0061 a 0328 ◌̨ |
| | | ~ | supported in Junicode Beta since Jun. 30, 2020: 0104 ą cv02[1] 󰀁 LATIN SMALL LETTER A WITH STROKE THROUGH TERMINAL |

**Figure 6**: Adapting output to tag variation sequences.

also be considered). How to process this information further is another question outside the scope of this note. One possible approach is to continue the work on the fntsample fork. Another possibility is to use XSLT to convert to LaTeX (running under XeTeX or LuaTeX); this is a general recommendation of David Carlisle, the author of xmltex.[23]

The Unicode standard is updated every year, so we need a way to handle the update conveniently. Conversion to XML with Parafiński's parser, after some minor improvements of the program, seems a satisfactory solution.

In the Private Use Area, the updates to Medieval Unicode Font Initiative recommendation, for example, don't have a stable specific form. The data, e.g., as prepared for Parafiński's program, have to be updated by hand.

One way to extract the interesting information from the font which seems to be worth consideration

[23] `tex.stackexchange.com/questions/562856`

```
Table: GSUB
  Script: DFLT
    Language: default
      Feature: aalt
        Lookups: 0,1
      Feature: c2sc
        Lookups: 204
...
      Feature: ccmp
        Lookups: 50,53,55,56,57,58,59,60,61,...
      Feature: cv01
        Lookups: 110
      Feature: cv02
        Lookups: 111
...
```

**Figure 7**: Some of the output from `layout-features.py JunicodeTwoBeta-Regular.ttf`

consists of using the `ttx` plain text format of OpenType/TrueType fonts produced by the program of the same name.[24]

To make the output more human readable, by default `ttx` uses the character names from the file `UnicodeData.txt` belonging to the Unicode Character Database. There is, however, an option to provide a different file for the data. One occasion for this is a new version of the standard which has not yet migrated to the relevant software libraries. The second, more important for us, is to provide names of the PUA characters, including MUFI, prepared already in the appropriate format by Rebecca G. Bettencourt[25] (updates are probably needed).

OpenType/TrueType fonts consist of several tables and subtables, which `ttx` converts to XML (in the examples below, the formatting is slightly changed). Variation and tag sequences are represented in the same way as other ligatures. For example, in `NotoSans-Regular.ttf` the slashed zero variation sequence (surprisingly few fonts support this variation sequence), stored in the `GSUB` (Glyph Substitution) table, looks like this:

```
<GSUB>
  <Version value="0x00010000"/>
  ...
  <LookupList>
    <!-- LookupCount=43 -->
    <Lookup index="2">
      ...
      <!-- SubTableCount=1 -->
      <LigatureSubst index="0" Format="1">
        <LigatureSet glyph="zero">
          <Ligature components="uniFE00"
                    glyph="zero.slash"/>
        </LigatureSet>
      </LigatureSubst>
    </Lookup>
    ...
  </LookupList>
</GSUB>
```

To discover what `uniFE00` definitively means, we consult the `cmap` (Character to Glyph Index Mapping) table:

```
<cmap>
  <tableVersion version="0"/>
  <cmap_format_4 platformID="0" platEncID="3"
                 language="0">
    ...
    <map code="0xfe00" name="uniFE00"/>
         <!-- VARIATION SELECTOR-1 -->
  </cmap_format_4>
  ...
</cmap>
```

The comments are provided by the `ttx` program.

As for `zero.slash`, we can use the auxiliary table `GlyphOrder` produced by `ttx` to find the font slot number of the character:

```
<GlyphOrder>
  <!-- The 'id' attribute is only for humans;
       it is ignored when parsed. -->
  ...
  <GlyphID id="2581" name="zero.slash"/>
  ...
</GlyphOrder>
```

The slot number can be used to typeset the character; in XƎTEX the appropriate command is `\XeTeXglyph`.[26]

Let's now consider another example, namely *b quadratum* from the Junicode font mentioned above. The tag sequence is active only when stylistic set 10 is selected, so the ligature has to be embedded in the appropriate `FeatureElement`:

```
<GSUB>
  <Version value="0x00010000"/>
  ...
  <FeatureList>
    <!-- FeatureCount=155 -->
    ...
    <FeatureRecord index="140">
      <FeatureTag value="ss10"/>
      <!-- Character Entities
           for Combining Marks -->
      <Feature>
        <FeatureParamsStylisticSet>
          <Version value="0"/>
```

[24] `github.com/fonttools/fonttools`; see also [4, p. 22].
[25] `kreativekorp.com/charset/PUADATA/PUBLIC/MUFI`

[26] This was kindly pointed out to me by Ulrike Fischer on the XƎTEX mailing list (`tug.org/pipermail/xetex/2022-October/028105.html`). [Editor's note: For LuaTEX, some Lua code can achieve the same result; see `tex.stackexchange.com/questions/120736`.]

```
        <UINameID value="256"/>
                  <!-- Entities -->
      </FeatureParamsStylisticSet>
      <!-- LookupCount=3 -->
      ...
      <LookupListIndex index="2"
                       dopiskivalue="71"/>
    </Feature>
  </FeatureRecord> ...
</FeatureList>
<LookupList>
  <!-- LookupCount=234 -->
  ...
  <Lookup index="71">
    <LookupType value="4"/>
    <LookupFlag value="16"/>
              <!-- useMarkFilteringSet -->
    <!-- SubTableCount=1 -->
    <LigatureSubst index="0" Format="1">
      ...
      <LigatureSet glyph="b">
        <Ligature components="e.tag,n.tag"
                  glyph="b.enlarged"/>
        <Ligature components="p.tag,l.tag"
                  glyph="b.p02"/>
        <Ligature components="p.tag,s.tag"
                  glyph="b.p01"/>
      </LigatureSet>
      ...
    </LigatureSubst>
    <MarkFilteringSet value="1"/>
  </Lookup> ...
</LookupList>
</GSUB>
```

In general the glyph selection can depend on a script, a language, and user-selected features, which make the font structure quite complicated. A program intended to extract all the information about a font has to take everything into account.

## 7 Final remark

I hope this note will provide inspiration to a reader or readers with appropriate skills and in some future we will see a tool for printing annotated font tables in a nice fashion.

## References

[1] P. Baker. Junicode — the font for medievalists. specimens and user manual for version 2, 2022. github.com/psb1558/Junicode-font/

[2] J.S. Bień. Representating Parkosz's alphabet in the Junicode font. *TUGboat* 43(3):247–251, 2022. tug.org/TUGboat/tb43-3/tb135bien-parkosz.pdf

[3] M. Davis, K. Whistler. Default ignorable issues. L2/02-368, 2002. unicode.org/L2/L2002/02368-default-ignorable.pdf

[4] Y. Haralambous. *Fonts & Encodings. From Advanced Typography to Unicode and Everything in Between.* O'Reilly Media, 2007.

◇ Janusz S. Bień
Warsaw, Poland
jsbien (at) uw.edu.pl
sites.google.com/view/jsbien
ORCID 0000-0001-5006-8183

## Production notes

Karl Berry

Almost all of the characters in Janusz's article could be typeset with no particular trouble. But two needed special attention: the character  (U+F0001 LATIN SMALL LETTER A WITH STROKE THROUGH TERMINAL) and the visible glyph ︀ (U+FE00 VARIATION SELECTOR-1).

For the former, XƎLaTeX has no problems typesetting U+F0001 from the Junicode font:

```
\newfontfamily{\Junicode}
  {JunicodeTwoBeta-Regular.ttf} % for XeTeX
\newcommand{\sgl}[1]
  {{\Junicode #1}}
\newcommand{\Fzerosone}{\sgl{...}}
```

However, I wanted to use LuaLaTeX to typeset the article, because its support for microtype's font expansion feature avoided several overfull lines, and it typeset some other character. It turns out (github.com/latex3/luaotfload/issues/244) that setting the HarfBuzz rendering mode is what's needed. (This is not the default in lualatex, even though it uses the luahbtex engine.)

```
% For LuaTeX:
\newfontfamily{\Junicode}
  [Renderer=HarfBuzz]{JunicodeTwoBeta-Regular.ttf}
```

For the latter character: ordinarily, Unicode prescribes that variation selectors are invisible, but a few fonts also provide a visible glyph; the one here (found by Janusz) is from NotoSansManichaean-Regular.ttf, following what is printed in the Unicode font charts.

XƎLaTeX could handle this with its \XeTeXglyph primitive, which can be used to typeset any glyph from a font, whether mapped to an input character or not; in this case, \XeTeXglyph 58. (The ttx program can be used to discern such internal information in any OpenType or TrueType font.)

For LuaLaTeX, however, it was necessary both to use the Base rendering mode, and a bit of Lua code devised by Henri Menke (thank you Henri, and thank you search engines), which emulates many XƎTeX primitives in LuaTeX (gist.github.com/hmenke/6e8ff7c90a5e5df3c4895f60059a2ef7):

```
\ifx\undefined\XeTeXglyph % LuaTeX case:
  \def\XeTeXglyph{%
    \directlua{...}}%
\fi
\newfontfamily{\NSM}
  [Renderer=Base]{NotoSansManichaean-Regular.ttf}
\newcommand{\VSone}{{\NSM\XeTeXglyph 58}}
```

Happy Unicode typesetting.

◇ Karl Berry
github.com/TeXUsersGroup

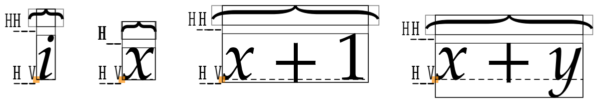
## OpenType extensible brace debugging

Hans Hagen, Mikael P. Sundqvist

When you combine writing a new math manual with development of the math typesetting subsystem, you can run into surprising buglets.

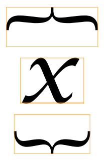
A valid traditional TEX approach to putting braces over or under (a bit of) formula is to assemble such a brace from five snippets, where the left, middle and right snippet are characters and the "even" ones are rules that can stretch. An OpenType math font can have line segments that are used instead of rules. But before that assembly happens, one can first check if there are precomposed wider variants in the font.

Consider the following formula elements (typeset with simply `$\overbrace{i}$`, etc.):

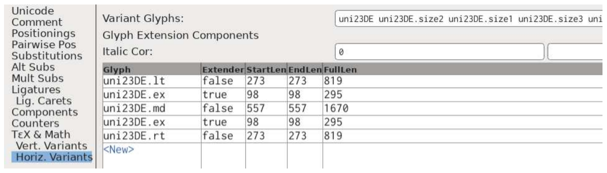


When you test examples like this you would expect the overbrace on the $i$ to be narrower than on the $x$. But in the above, using Pagella (its design makes the problem apparent), we see they are the same size. Also, the braces above $x + 1$ and $x + y$ are slightly wider than they need to be.

Another issue is that we expect braces above and below (with `\underbrace`) to have the same size. Switching to Lucida for the example below, we can see that they don't; the brace on top is noticeably bigger:



Of course you will seldom put a brace both on top of and below a single character, and that is likely why this went unnoticed for quite a while. And, because normally code involved in handling this is kind of symmetric, one starts wondering about the font. And indeed, when we looked into the font, we found that `uni23DE` (TOP CURLY BRACKET) had an error in the variant list: `.size1` and `.size2` were swapped. In FontForge (the variant list is given in the box at the upper right of the screenshot):



Variant Glyphs: uni23DE uni23DE.size2 uni23DE.size1 uni23DE.size3 uni

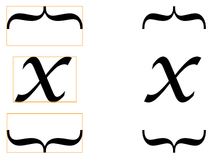
Glyph Extension Components

Italic Cor: 0

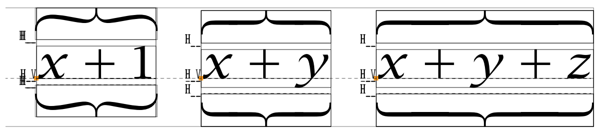
| Glyph | Extender | StartLen | EndLen | FullLen |
|---|---|---|---|---|
| uni23DE.lt | false | 273 | 273 | 819 |
| uni23DE.ex | true | 98 | 98 | 295 |
| uni23DE.md | false | 557 | 557 | 1670 |
| uni23DE.ex | true | 98 | 98 | 295 |
| uni23DE.rt | false | 273 | 273 | 819 |

Because we had just updated this font[1] we immediately thought that we had messed up, but a look at the old version showed us that this swap was already there: it just went unnoticed! The simple fact that we showed both braces made it prominent. How likely is it to have two braces on a single character?

Sorting the sizes correctly results in braces of the same size (left below), but you may notice that the braces are still wider than the $x$. We can fix this by scaling the brace slightly (right below).

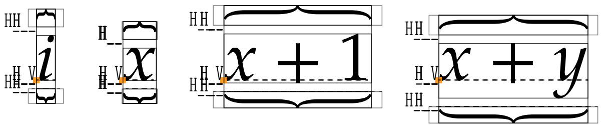


While looking closer at this, we realized that another problem exists: Lucida's widest precomposed brace character is much smaller than the narrowest extensible brace. Also, the precomposed braces are naturally "curvier" than the extensible ones, so the braces for $x+1$ are quite different compared to $x+y$:



Since the different curviness is by design it is nothing to fix, but in a future version we might need to add a size or modify the extensible recipe.

Since the problems have gone unnoticed, there is no hurry to push another release, though. Meanwhile, in ConTEXt, we can easily fix the swap with a tweak in the goodie file (by sorting variants on size), and users will not notice the gap between the sizes, since we stretch or shrink the result. Here we show how it looks for Pagella, to be able to compare with the first figure, with stretching and shrinking enabled:



Note that the sizes of all braces now match their content.

Font debugging grows ever more complex ...

◇ Hans Hagen
Pragma ADE

◇ Mikael P. Sundqvist
Department of Mathematics
Lund University
`mickep (at) gmail dot com`

[1] `tug.org/TUGboat/tb43-3/tb135hagen-lucida.pdf`

## ConTEXt in TEX Live 2023

Hans Hagen

Starting with TEX Live 2023 the default ConTEXt distribution is LMTX, a follow up on MkIV, running on top of the LuaMetaTEX engine instead of LuaTEX. Already for a long time the MkII version used with pdfTEX, XƎTEX and Aleph has been frozen and most users moved on from MkIV to LMTX (a more distinctive tag for what internally is version MkXL).

In principle one can argue that we now have three versions of ConTEXt and there can be the impression that they are very different. However, although MkXL can do more than MkIV which can do more than MkII, the user interface hasn't changed that much and old functionality is available in newer versions. Of course some old features make no sense in newer variants, like eight-bit font encodings in an OpenType font realm and input encodings when one uses UTF, although we still support input encodings a.k.a. regimes. When we started using the Mk`*` suffixes the main reason was that we had to distinguish files and the official TEX distribution doesn't permit duplicate file names. Using a distinctive suffix also makes it possible to treat files differently.

Table 1 shows major aspects of the different ConTEXt versions. The 'template' files listed in the table are a mix of TEX and Lua and originate in the early days of MkIV; basically, they are a wink to active server pages. With 'arguments', we refer to files that accept named macro arguments which means that they need to be preprocessed. That started as a proof of concept but some core files are defined that way. Users will normally just use a `.tex` file.

The Lua files in the code base have the suffix `lua`, or when meant for LuaMetaTEX that uses a newer Lua engine they can have the suffix `lmt`. There can also be `lfg` (font goodies) and `llg` (language goodies) plus byte-compiled files with various suffixes but these are normally not seen by users. We leave it at that.

So, while TEX Live 2022 installed MkII and MkIV, TEX Live 2023 installs MkIV and LMTX. Therefore the most significant upgrade is in the engine that is used by default: LuaMetaTEX instead of LuaTEX. The MkII files are no longer installed so we don't need pdfTEX.

So how did we end up here? Initially the idea was that, because LuaTEX is basically frozen, LuaMetaTEX would be the engine that we conduct experiments with and from which occasionally we could backport code to LuaTEX. However it soon became clear that this would not work out well so backporting is off the table now. Just for the record: the project started years ago so we're not talking about something experimental here. There have been articles in *TUGboat* about what we've been doing over the years.

One of the first decisions I made when starting with LuaMetaTEX was to remove the built-in backend, which then meant also removing the bitmap image inclusion code. That made us get rid of dependencies on external libraries. In fact, a proof-of-concept experimental variant didn't use the built-in backend at all. The font loading code could be removed as well because that was not used in MkIV either. In MkIV we also don't use the `kpse` library for managing files so that code could be dropped from the engine tool; it can be loaded as so-called optional library if needed but I'll not discuss that here. If you look at what happens with the LuaTEX code base, you'll notice that updating libraries happens frequently and that is not a burden that we want to impose on users, especially because it also can involve updating build-related files. Another advantage of not using them is that the code base remains small.

A direct consequence of all this was that the build process became much more efficient and less complex. A fast compilation (seconds instead of minutes) meant that more drastic experiments became possible, like most recently an upgrade of the math subsystem. All this, combined with an overhaul of

| suffix | engine | template | arguments | main file |
|---|---|---|---|---|
| MkII | pdfTEX, XƎTEX, Aleph | | | `context.mkii` |
| MkIV | LuaTEX, LuaJITTEX, LuaMetaTEX | | | `context.mkiv` |
| MkVI | idem | | yes | |
| MkIX | idem | yes | | |
| MkXI | idem | yes | yes | |
| MkXL | LuaMetaTEX | | | `context.mkxl` |
| MkLX | idem | | yes | |

**Table 1**: Major ConTEXt versions.

the code base, both the TEX and MetaPost part, meant that backporting was no longer reasonable. Being freed from the constraint that other macro packages might use LuaMetaTEX in turn resulted in more drastic experiments and adding features that had been on our wish list for decades. Another side effect was that we could easily compile native Windows binaries and immediately support transitions to ARM-based hardware.

Instead of "backporting after experimenting", a leading motive became "fundamentally move forward" while at the same time tightening the relation between ConTEXt and the engine: the engine code became part of the distribution so that users can compile themselves, which fits perfectly in the paradigm (and demands) of distributing all the source code, even that of the engine. There is also less danger that patches on behalf of other usage interferes with stable support for ConTEXt. A specific installation is now more or less long-term stable by design because it no longer depends on binaries and/or libraries being provided for a specific platform and operating system version. Of course installers and TEX Live do provide the binaries, so users aren't forced to worry about it, but they can move along with a system update by recompiling an old, and for their purpose, frozen ConTEXt code base.

An unofficial objective (or challenge) became that the accumulated source stays around 12 MB uncompressed, (compressed a bit over 2 MB) and the binary around 3 MB so that we could use the engine as an efficient Lua runner as well as a launcher stub, thereby removing yet another dependency. That way the official ConTEXt distribution didn't grow much in size. A bonus is that we now use the same setup for all operating systems. It also opened up the possibility of a exceptionally small installation with all bells and whistles included. Another nice side effect, combined with automatic compilation on the compile farm, makes that we can provide installations that reflect the latest state of affairs: a recent binary combined with the latest ConTEXt. As a result, most users quickly went for LMTX instead of MkIV.

In the code base we avoid dependencies on specific platforms but there are a few cases where the code for Windows and UNIX differs. However, the functionality should be the same. A good test is that for Windows we can compile with mingw (cross-compilation), MSVC (native) and clang (native); that order is also the order of runtime performance. The native MSVC binary is the smallest but users probably don't care. In any case, it is nice to have a fallback plan in place. The code is all in C; the MetaPost code is converted from CWEB into C using a Lua script but we also ship the resulting C code. The code base provides a couple of CMake files and comes with a trivial build script.

When I say that there are no libraries used, I mean external libraries. We do use code from elsewhere: adapted `avl` as well as `decnumber` (for the MetaPost library), adapted `hjn` (hyphenation), `miniz` (zip compression), `pplib` (for loading PDF files), `libcerf` (to complement other math library support, but it might be dropped), and `mimalloc` for memory management. However all the code is in the LuaMetaTEX code base and only updated after checking what changed. The most important library originating elsewhere is of course Lua: we use the latest and greatest (currently) 5.4 release. We kept the `socket` library but it might be dropped or replaced at some point. In addition there is a subsystem for dynamically loading libraries; the main reason for that being that I needed `zint` for barcodes, interfaces to sql databases, a bunch of compression libraries, etc. But all that is tagged *optional* and ConTEXt will never depend on it. There are no consequences for compilation either because we don't need the header files. The glue code is very minimalistic and most work gets delegated to Lua.

Initially, because the backend is written in Lua, there was a drop in performance of some 15% but that was stepwise compensated by gains in performance in the engine and additional or improved functionality. The ConTEXt code base is rather optimized so there was little to gain there, apart from using new features. Existing primitive support could also be done a bit more efficiently; it helps if one knows where potential bottlenecks are. Therefore, in the meantime an LMTX run can be quite a bit faster than a MkIV run and it can even outperform a LuaJITTEX run. In practice, the difference between an eight-bit MkII run using the eight-bit pdfTEX engine and a 32-bit LuaMetaTEX run with LMTX can be neglected, definitely on more complex documents. I never get complaints about performance from ConTEXt users, so it might be a minor concern.

So what are the main differences in the installation? If you really want to experience it you should use the standard installation. Currently the small installer is the engine that synchronizes the installation over the net and, assuming a reasonable internet connection, that takes little time. The installation is relatively small, and many of the bytes used are for the documentation. Updates are done by transferring only the changed files. The TEX Live installation is a bit larger because it shares for instance fonts with the main installation and these come with resources used by other macro packages. Both installations bring MkIV as well as LMTX and therefore provide LuaTEX as well as LuaMetaTEX. However, a MkIV run is now managed by LuaMetaTEX because we use that engine for the runner. The MkII code is no longer in TEX Live but is in the repositories and used to test and compare with pdfTEX. It just works.

The number of binaries and stubs is reduced to a minimum:

| | |
|---|---|
| `luametatex` | combined TEX, MetaPost, Lua engines |
| `mtxrun` | script runner, binary |
| `context` | ConTEXt runner, binary |
| `mtxrun.lua` | script runner, Lua code |
| `context.lua` | loader for ConTEXt runner |
| `luatex` | the good old ancestor |

All of these programs are in the ConTEXt distribution directory `tex/texmf-`⟨*platform*⟩`/`. In addition, `context` and `mtxrun` are symlinks to the `luametatex` binary, where possible.

So, the `context` command runs `luametatex`, but loads the Lua file with the same name which in turn will locate the ConTEXt management script (`mtx-context`) in the TEX tree and run it. The same is true for `mtxrun`: it is a binary (link) that loads the script in (this time) the same path and then can perform numerous tasks. For instance, identifying the installed fonts so that they can be accessed by name is done with:

```
mtxrun --script font --reload
```

Where in MkII we had stubs for various utility scripts, already in MkIV we went for a generic runner and a bit more keying. It's not like these scripts are used a lot and by avoiding shortcuts there is also little danger for a mixup with the ever-growing list of other scripts in TEX Live or commands that the operating system provides.

The LuaTEX binary is optional and only needed if a user also wants to process MkIV files. There are no shell scripts used for launching. The two main calls used by users are:

```
context foo.tex
context --luatex foo.tex
```

A user has only to make sure that the binaries are in the path specification. When you run from an editor, the next command does the work:

```
mtxrun --autogenerate --script context ⟨filename⟩
```

with ⟨*filename*⟩ being an editor-specific placeholder. Like other engines, LuaMetaTEX (and ConTEXt) needs a file database and format file, and although it should generate these automatically you can make them with:

```
mtxrun --generate
context --make
```

The rest of the installation is similar to what we always had and is TDS compliant. The source code of LuaMetaTEX is included in the distribution itself (which nicely fulfills the requirements) but can also be found at `github.com/contextgarden/luametatex`.

There are also some optional libraries there but ConTEXt works fine without them. The official latest distribution of ConTEXt itself is:

```
github.com/contextgarden/context
github.com/contextgarden/context-distribution-fonts
```

We see users grab fonts from the Internet and play with them. They can install additional fonts in `tex/texmf-fonts/data/`⟨*vendor*⟩. Project-specific files can be collected in `tex/texmf-project/tex/context/user/`⟨*project*⟩. These directories are not touched by installations and can easily be copied or shared between different installations. After adding files to the tree `mtxrun --generate` will update the file database.

In the distribution there are plenty of documents that describe how LuaMetaTEX with LMTX differs from MkIV with LuaTEX: new primitives, macro extensions, more granular math rendering, improved memory management, new (or extended) (rendering) concepts, more MetaPost features; most is covered in one way or another, and much is already applied in the ConTEXt source code. After all, it took a few years before we arrived here so you can expect substantial refactoring of the engine as well as the code base, and therefore eventually there is (and will be) more than in MkIV.

When you compare a ConTEXt installation with what is needed for other macro packages you will notice a few differences. One concerns the way TEX is launched. An engine starts with a blank slate but can be populated with a so-called format file that is basically a memory dump of a preloaded macro package. So, the original way to process a file is to pass a format filename to the engine. In order to avoid that a trick is used: when an engine (or symlink/stub to it) is launched by its format name, the loading happens automatically. So, for instance `pdflatex` is actually an equivalent for starting pdfTEX with the format file `pdflatex.fmt` while `latex` is pdfTEX with another format file (`latex.fmt`) starting up in DVI mode. And, as there are many engines, a specific macro package can have many such combinations of its name and engine.

In ConTEXt we don't do it that way. One reason is that we never distinguished between backends: MkII uses an abstract backend layer and load driver files at runtime (it was one of the reasons why we could support Acrobat as soon as it showed up, because we already supported the now obsolete but quite nice DVIWINDO viewer). And that model hasn't changed much as we moved on. Because we use a runner, we also don't need to distinguish between engines: all formats have the same name but sit on an engine subpath in the TEX tree. Anyway, this already removes quite some formats. On the other hand, ConTEXt can be run with different language specific user interfaces which means that instead of just `context.fmt` we have `cont-en.fmt` and possibly more, like `cont-nl.fmt`. So that can increase the number again but by default only the English interface is installed. As a side note: where with MkII we needed to generate MetaPost mem files, with its descendants having MPlib we load the (actually quite a bit of) MetaPost code at runtime.[1]

In addition to a format file, for the LuaTEX and LuaMetaTEX engine we also have a (small) Lua loader alongside the format file. All this is handled by the runner, also because we provide extensive command line features, and therefore of no concern to users and package maintainers. However, it does make integrating ConTEXt in for instance TEX Live different from other macro packages and thereby puts an extra burden on the TEX Live team. Here I want to thank the team for making it possible to move forward this way, in spite of this rather different approach. Hopefully a LuaMetaTEX integration is a bit easier in the long run because we no longer have different stubs per platform and at least the binary part now has no dependencies and only has a handful of files.

For those new to ConTEXt or those who want to try it in TEX Live 2023 there is not much difference between the versions. However, MkIV is now frozen and new functionality only gets added to LMTX. Of course we could backport some but with most users already having moved on, it makes no sense. Just as we keep MkII around for testing with pdfTEX, we also keep MkIV alive for testing with LuaTEX. Maybe in a couple of years MkIV will go the same route as MkII: ending up in the archives as an optional installation.

◇ Hans Hagen
Pragma ADE

[1] Occasionally I do experiments with loading the TEX format code at runtime, but at this moment the difference in startup time of about one second (assuming files are cached) is too large and running over networks will be less fun, so the format file will stay. The time involved in loading MetaPost can be brought down but for now I leave it as it is.

## Preserving the math class of variables

Hans Hagen

If there is one thing that OpenType math has made clear, it's that we have lots of alphabets. It is customary in a TEX document to key in regular (ASCII) letters and expect them to become for instance math italic, bold upright, script or whatever.

One way to do this is to relate a character (directly or by name) to a specific slot in a font assigned to a so-called math family, which groups text, script and scriptscript sizes. Here are a couple ways to do this, using the Unicode `\Umathcode` primitive:

```
\Umathcode‘a = "0 "9 ‘a
\Umathcode‘a = "0 "5 "1D44E
```

In the first line we map the input character `a` (the first `‘a`) to the glyph slot of `‘a` (the second one; that is, 97) in family 9. In the second line, the input `a` is mapped to the Unicode math italic alphabet's $a$, using family 5. The `"0` in both lines is the math class, in this case specifying an "ordinary" character.

Switching families can be done directly, although more usually it is wrapped in a command:

```
$ a + {\fam"9 a} + {\fam"5 a} $
```

For our next example, we take a colon from family zero (`"0`) and assign it class 6 (`"6`) which means that it will get punctuation spacing (like `\Colon`):

```
\Umathchardef\foo "6 "0 ‘: % punct
```

In the following line we do the same but with class 7, which is "variable", meaning TEX uses the current family, as stored in the `\fam` primitive parameter.

```
\Umathchardef\foo "7 "0 ‘: % ord
```

Doing this, we lose the prior class value (3), so we end up with ordinary (which normally means no) spacing. In LuaTEX (>1.15.1) we can now preserve the class by declaring and using a special "variable" family instead:

```
\variablefam"24
\Umathchardef\foo "6 "24 123 % punct
```

When a character has family `\variablefam` assigned, it will get the current `\fam` value and the class can remain 3, as specified.

This is a relatively cheap extension which we prototyped in LuaMetaTEX and backported to LuaTEX. We don't use this in ConTEXt (just to warn its users) but it might be handy in other macro packages.

◇ Hans Hagen
Pragma ADE

doi.org/10.47397/tb/44-1/tb136hagen-classes

# Creating macros in OpTeX

Petr Olšák

## Introduction

OpTeX [1] is an extended plain TeX. We can create macros as in plain TeX. In particular, this means that we use TeX primitives like `\def`, `\edef`, `\ifx`, `\expandafter`, `\csname`, `\hbox`, `\vbox`, `\hrule`, and so on. Likewise, we use basic plain TeX macros like `\newcount`, `\llap` and many others. I wrote a summary of these TeX and plain TeX tools in [2].

OpTeX keeps the plain TeX philosophy: it does not create any new syntactic, semantic, or thought layers over TeX, so the commands mentioned above are principal ones, basic for creating macros. For example, OpTeX doesn't try to provide anything similar to `\newcommand`, nor anything similar to `expl3`. The main message is: if you know TeX, you can make your macros.

On the other hand, OpTeX provides many elementary macros which can make macro programming easier. And there are a few conceptual recommendations especially to separate different namespaces when your macros will be used for public purposes. This article summarizes these tools and principles. More detailed information can be found in the OpTeX manual [3].

## Naming conventions and namespaces

When you are creating macros for your use then you can use arbitrary alphabetical names for newly declared control sequences. Moreover, you can redefine existing names, if you decide that it is useful and you never are using them in their original meaning. For example, you can define `\def\box{...}` without any problem, as long as you use it only with your declared meaning. It doesn't matter that `\box` is a TeX primitive in its original meaning. OpTeX internally uses copies of all primitive names and internal macro names, `\_box` in this case.

In other words, when OpTeX starts, all internal sequences are duplicated (both `\box` and `\_box` are present, with the same meaning) and OpTeX uses only the name `\_box` in its internal macros. A user can redefine `\box` if he or she finds it useful, or doesn't know that the name is already used. There is only one requirement: if you re-declare a control sequence then it cannot be used in its original meaning in your document. For example, `\def\def{...}` is possible but then you cannot use `\def` as a primitive command in your next text.

The alphabetical control sequences like `\foo`, `\SomethingOther`, `\hbox` are considered in "public namespace" from the OpTeX point of view. On the other hand, alphabetical control sequences beginning with "`_`" (like `\_foo`) are reserved in special namespaces. The character "`_`" has category code 11 (letter) in OpTeX.* This means that you have full access to the internal control sequences like `\_foo` without any category dancing. You don't need to say something like `\makeatletter ... \makeatother`. You can use these internal sequences in "read-only" mode without any restrictions. You can redefine them too, but if you decide to do that then you hopefully know what you are doing, and what internal process in OpTeX will be changed.

As mentioned above, "OpTeX's private namespace" includes names beginning with "`_`" followed by normal letters. There are copies of all TeX primitives and internal OpTeX macros here. An internal macro of a macro package uses its "package's private namespace" `\_pkg_foo`, where "`pkg`" is a shortcut of the package. A package writer uses the `\_namespace` declaration for dealing with such control sequences more comfortably; see below, the section "writing public macro packages".

The single-letter control sequences (like `\$`, `\/`, `\,`) are declared similarly to plain TeX** and are not used in internal OpTeX macros. Users can re-declare them freely without affecting the behavior of OpTeX.

## Basic macros for macro programmers

OpTeX provides a few basic macros:

- `\sdef{`⟨*cs-name*⟩`}` defines a control sequence whose name is given by the ⟨*cs-name*⟩ string. Thus, `\sdef {T\the\mycount}#1:#2{...}` is (roughly speaking) an equivalent to `\def\T42 #1:#2{...}` if `\mycount=42`.
- `\sxdef{`⟨*cs-name*⟩`}` is similar to `\sdef`, but the `\xdef` primitive is used behind the scenes instead of `\def`.
- `\slet{`⟨*cs-name1*⟩`}{`⟨*cs-name2*⟩`}` is (roughly speaking) equivalent to `\let` ⟨*cs-name1*⟩`=` ⟨*cs-name2*⟩.

* There is a little trick to enable use of this character with its normal plain TeX meaning in math mode without changing this category. But it works.

** Not all single letter control sequences from plain TeX are available. Control sequences for accents like `\"`, `\'` are undefined by default because we suppose that accented letters are directly written in Unicode (the current year is 2023). But a conservative user can enable them with the `\oldaccents` declaration.

doi.org/10.47397/tb/44-1/tb136olsak-optexmac

- `\adef` ⟨*character*⟩ sets the ⟨*character*⟩ to be active and defines it like `\def`⟨*character*⟩. For example, `\adef *{...}` or `\adef @#1#2{...}`.
- `\optdef\macro [`⟨*default*⟩`]`⟨*params*⟩ is similar to `\def\macro` ⟨*params*⟩ but the `\macro` can be used with an optional argument given like `\macro[`⟨*text*⟩`]` before scanning other parameters. If the optional syntax is used then the token register `\opt` includes ⟨*text*⟩. Otherwise, it includes ⟨*default*⟩.
- `\eoldef\macro #1{...}` defines `\macro` with a single parameter delimited by the end of the current line. This parameter is `#1`, and it can be used in the macro body. Such a `\macro` can be used only when lines of text are read, not inside other macros.
- `\cs{`⟨*text*⟩`}` is a shortcut for the commonly-used `\csname` ⟨*text*⟩`\endcsname`.
- `\trycs{`⟨*text*⟩`}{`⟨*else*⟩`}` does `\cs{`⟨*text*⟩`}` only if `\cs` is defined, otherwise the ⟨*else*⟩ part is processed.

There is a useful shortcut of the `\expandafter` primitive: `\ea`. Of course, it is safer to use `\_ea` because control sequences with short names tend to be re-declared later by a user.

Additional simple macros include:

- `\ignoreit{`⟨*text*⟩`}` does nothing.
- `\useit{`⟨*text*⟩`}` does ⟨*text*⟩.
- `\ignoresecond{`⟨*A*⟩`}{`⟨*B*⟩`}` does ⟨*A*⟩.
- `\usesecond{`⟨*A*⟩`}{`⟨*B*⟩`}` does ⟨*B*⟩.

You can add a given text to a parameterless macro:

- `\addto\macro{`⟨*text*⟩`}` appends ⟨*text*⟩ to the `\macro` body.
- `\aheadto\macro{`⟨*text*⟩`}` prepends ⟨*text*⟩ to the `\macro` body.

You can globally increase a counter by one by `\incr`⟨*counter*⟩ and decrease it by `\decr`⟨*counter*⟩. `\opwarning{`⟨*message*⟩`}` prints a message to the terminal and log file.

## Branching macro processing

Of course, you can use all of TeX's `\if*` primitives. OpTeX also provides the following `\is*` conditionals with the general syntax being one of:

`\isfoo...\iftrue` ⟨*true-text*⟩`\else` ⟨*false-text*⟩`\fi`
or
`\isfoo...\iffalse` ⟨*false-text*⟩`\else` ⟨*true-text*⟩`\fi`

The macro `\isfoo` calculates the condition and gobbles the `\iftrue` or `\iffalse`. You have to use this syntax because the `\isfoo` block can be skipped by another outer `\if` condition and the pairs `\if`...`\fi` must match.

The `\isfoo` macros are:

- `\isempty{`⟨*text*⟩`}\iftrue` is true if ⟨*text*⟩ is empty.
- `\isequal{`⟨*text-A*⟩`}{`⟨*text-B*⟩`}\iftrue` is true if the string ⟨*text-A*⟩ is equal to ⟨*text-B*⟩. These parameters are treated as strings. The category code of the characters has no effect.
- `\ismacro\macro{`⟨*text*⟩`}\iftrue` is true if the body of the parameterless `\macro` is equal to given ⟨*text*⟩.
- `\isdefined{`⟨*cs-name*⟩`}\iftrue` is true if the ⟨*cs-name*⟩ is defined.
- `\isinlist\list{`⟨*text*⟩`}\iftrue` is true if the ⟨*text*⟩ is included in the `\list` macro body.
- `\isfile{`⟨*file-name*⟩`}\iftrue` is true if the file named ⟨*file-name*⟩ exists and is accessible by TeX for reading.
- `\isfont{`⟨*font-name*⟩`}\iftrue` is true if the font named ⟨*font-name*⟩ exists. You can use `[`⟨*font-file*⟩`]` instead of ⟨*font-name*⟩ too.

All these `\is`-macros are fully expandable. The following `\is`-macro has different syntax than the macros mentioned above, but it is also expandable:

- `\isnextchar` ⟨*char*⟩`{`⟨*true-text*⟩`}{`⟨*false-text*⟩`}`. If the next character is equal to ⟨*char*⟩ then ⟨*true-text*⟩ is processed else ⟨*false-text*⟩ is processed.

OpTeX provides the `\afterfi{`⟨*text*⟩`}` macro which can be used inside `\if`...`\else`...`\fi`. The macro closes the `\if`...`\fi` block and runs ⟨*text*⟩ after it is closed. For example

```
\ifx\a\b ... \afterfi{do something}%
   \else ... \afterfi{do something else}%
\fi
```

This is almost the same as `\_ea` ⟨*token*⟩`\else` or `\_ea` ⟨*token*⟩`\fi` but the `\afterfi` parameter can include more than a single token.

A nested `\if`...`\fi` block can be inside the `\afterfi` parameter and `\afterfi` macros can be here too. It means that nested `\afterfi` macros work as expected. You don't need to escape from a nested `\if`...`\fi` block by a larger number of `\expandafter`s.

We must recall that usage of primitive conditionals with `\if`...`\fi` blocks hides one potential problem: if you are designing a macro that reads a TeX macro code token by token then your `#1` might be `\if` or `\else` or something similar. Usage of such `#1` inside your `\if`...`\fi` block in your macro causes TeX to give an error. What can you do in such a case? Here's one example, when looking for a specific token (~, here):

```
\ifx ~#1\ea\ignoresecond\else \ea\ignorefirst\fi
   {⟨true text with #1, we know that #1 is ~⟩}
   {⟨else text with #1⟩}%
```

## Branching with more structural macros

OpTEX provides macros \caseof and \xcaseof to switch among more alternatives. Usage of \caseof:

```
\caseof ⟨token⟩
   ⟨token A⟩ {⟨text A⟩}
   ⟨token B⟩ {⟨text B⟩}
   ⟨token C⟩ {⟨text C⟩}
   ...
   \_finc   {⟨else text⟩}%
```

If ⟨*token*⟩ is ⟨*token A*⟩ then only ⟨*text A*⟩ is processed, if ⟨*token*⟩ is ⟨*token B*⟩ then only ⟨*text B*⟩ is processed, etc. If ⟨*token*⟩ differs from all declared tokens, then ⟨*else text*⟩ is processed.

The \xcaseof macro is similar, but you can specify an arbitrary primitive \if-test instead of a token comparison only. The fragment of the code above with one condition more can be written as

```
\let\next=⟨token⟩
\xcaseof
   {\ifx\next A}       {⟨text A⟩}
   {\ifx\next B}       {⟨text B⟩}
   {\ifx\next C}       {⟨text C⟩}
   {\ifnum\mynum=12 } {⟨text 12⟩}
   ...
   \_finc {⟨else text⟩}%
```

A \caseof block is skippable by an outer \if...\fi block but \xcaseof is not.

If there is more than one "true" result of the conditions given by \xcaseof, then the first condition wins and the others are skipped.

The _finc separator followed by {⟨*else text*⟩} is obligatory. Of course, you can declare empty ⟨*else text*⟩. The separator must be written as _finc separator, not \finc. The reason is that the same syntax is given for _caseof and _xcaseof macros.

The spaces between \caseof or \xcaseof parameters are ignored but not the last space after the {⟨*else text*⟩}. Note the percent character in the examples.

\caseof and \xcaseof are fully expandable macros.

## Loops

You can use the classical plain TEX \loop macro. The one difference from plain's \loop is that OpTEX allows you to declare \if...\else...\repeat. But nothing more. There are still limitations here: \loop is not expandable and \loop inside \loop is possible only if the inner \loop is in a group.

OpTEX provides two additional looping macros, \foreach and \fornum. They are fully expandable and can be arbitrarily nested without declaring a group. The body of these macros is processed without opening and closing a group. The syntax of the \foreach macro is one of:

```
\foreach ⟨text⟩\do {⟨body⟩}
 or
\foreach ⟨text⟩\do ⟨parameter-spec⟩{⟨body⟩}
```

The first variant runs ⟨*body*⟩ for each token* from the ⟨*text*⟩. The current token (current parameter) can be processed inside the ⟨*body*⟩ as #1. If the \foreach block is included inside another macro then you have to use ##1; if it is inside a macro in a macro, or inside another \foreach or \fornum body, then use ####1, etc.

The second variant with ⟨*parameter-spec*⟩ enables scanning of the given ⟨*text*⟩ with an arbitrary parameter specification like with \def. You can declare separators for these parameters. For example, suppose we are creating a macro \macro which gets a parameter as a list of pairs in parentheses:

```
\macro{(a,b); (c,d); (1,42)}
```

and we want to read this and print these pairs in reverse order and with a different format: b/a d/c 42/1. We can do this by:

```
\def\macro#1{%
   \foreach #1\do ##1(##2,##3){##3/##2 }%
}
```

The unused ##1 is there because we want to ignore an optional "; " before the opening (.

This \macro is expandable, so you can use it inside the parameter of the \message primitive, for example.

The \fornum and \fornumstep macros have the following syntax:

```
\fornum ⟨from⟩..⟨to⟩ \do {⟨body⟩}
 or
\fornumstep ⟨step⟩: ⟨from⟩..⟨to⟩ \do {⟨body⟩}
```

The ⟨*body*⟩ is repeated for numbers starting at ⟨*from*⟩ and ending at ⟨*to*⟩. The \fornum increments the number by one. The second case uses the given ⟨*step*⟩. The parameters ⟨*from*⟩, ⟨*to*⟩, ⟨*step*⟩ can be

* Not always a single token: if the ⟨*text*⟩ includes {...} then all tokens inside these braces are taken at once, similar to the scanning of a macro parameter.

any arbitrary expression accepted by the `\numexpr` primitive. The current number is accessible in ⟨*body*⟩ as `#1` (or `##1` inside macros, etc.).

You may notice that there is a name conflict: the same control sequence `\foreach` is used by the Ti*k*Z package with different syntax and different features. OpTEX enables loading Ti*k*Z by `\load[tikz]`. If this is done then the `\foreach` from Ti*k*Z is available only within the `\tikzpicture`...`\endtikzpicture` environment. Outside this environment, the `\foreach` from OpTEX is active. Moreover, your macro code can use the private `\_foreach` from OpTEX if you want to be sure what you are using. With `\_foreach`, you have to use the `\_do` separator, instead of `\do`.

Why is there such a naming conflict? My macros are several decades old; older than Ti*k*Z. I don't want to rename this control sequence only due to Ti*k*Z (especially when I personally hardly use Ti*k*Z).

### Key–value syntax for parameters

Calling `\readkv{`⟨*list*⟩`}` or `\readkv\list` reads a given ⟨*list*⟩ or a `\list` macro in ⟨*key*⟩=⟨*value*⟩ format. These pairs are comma-separated, and the =⟨*value*⟩ may be missing. Once the ⟨*list*⟩ is read, you can access the ⟨*value*⟩ by expandable macro `\kv{`⟨*key*⟩`}`. If you only need to know whether the ⟨*key*⟩ was used then `\iskv{`⟨*key*⟩`}\iftrue` returns the answer.

You can also declare ⟨*code*⟩ to be processed whenever a particular ⟨*key*⟩ is encountered during `\readkv`. This is done with `\kvx{`⟨*key*⟩`}{`⟨*code*⟩`}`. The ⟨*code*⟩ can access the scanned ⟨*value*⟩ as `#1`. Specifying `\nokvx{`⟨*other code*⟩`}` declares common ⟨*other code*⟩ to process for all ⟨*keys*⟩ not declared by `\kvx`. The ⟨*other code*⟩ can use `#1` to access the ⟨*key*⟩ and `#2` to access the ⟨*value*⟩.

The ⟨*key*⟩=⟨*value*⟩ data are stored in and read from a dictionary with the name `\kvdict`, which is a token register. It is empty by default, i. e. the default dictionary has an empty name. You can manage more dictionaries by changing it.

The following example is borrowed from the OpTEX documentation. We define a macro `\myframe` which can scan optional parameters in `[...]` key–value format and sets colors and dimensions by these parameters. When we use, for example

```
\myframe [rule-width=2pt, frame-color=\Blue]
         {text}
```

then a frame around the given `text` with rule width 2pt in blue color is created. The macro can be defined like this:

```
\def\myframedefaults{% defaults:
   frame-color=\Black, % color of rules
   text-color=\Black,  % color of the text
   rule-width=0.4pt,   % width of rules
   margins=2pt, % space between text and rules
}
\optdef\myframe []#1{%
   \bgroup
   \readkv\myframedefaults \readkv{\the\opt}%
   \rulewidth=\kv{rule-width}%
   \hhkern=\kv{margins}%
   \vvkern=\kv{margins}\relax
   \kv{frame-color}%
   \frame{\kv{text-color}\strut #1}%
   \egroup
}
```

The `\myframe` macro from this example runs the `\frame` macro provided by OpTEX. Its parameters `\rulewidth`, `\hhkern` and `\vvkern` are set from values given in key–value format when `\myframe` is used. The `\myframedefaults` macro clearly specifies the default values and any user-given values are read from the optional argument from the `\opt` tokens register. The `\readkv` macro is used twice: first the default values are read and second, the user-specified values are read. The last assignment wins.

### Expressions

In addition to the well-known `\numexpr` primitive from $\varepsilon$-TEX, OpTEX provides the expandable macro `\expr{`⟨*expression*⟩`}`, which calls the Lua interpreter (OpTEX always runs under LuaTEX) and does arithmetic with decimal numbers. The number of decimal digits of the result is 3 by default; this can be overridden with the optional argument, as in `\expr[`⟨*digits*⟩`]{`⟨*expression*⟩`}`. Examples:

```
\expr{2*(4-1.3)}      % 5.400
\expr{math.sqrt(1/3)} % 0.577
\expr[14]{math.pi}    % 3.14159265358979
```

The OpTEX macro `\bp{`⟨*dimen*⟩`}` provides an expandable conversion of ⟨*dimen*⟩ to the decimal number which expresses the given value in `bp` units. The ⟨*dimen*⟩ can be an arbitrary expression accepted by the `\dimexpr` primitive. The result is a decimal number without a unit. It can be used in, for example, arguments to the `\pdfliteral` literal where we are using such numbers without units in low-level PDF commands. For example, `\bp{\parindent}` returns `19.925` in this document, because `\parindent` is set to 20pt here.

You can use the `\bp{`⟨*dimen*⟩`}` as operands in the `\expr{`⟨*expression*⟩`}`. This is very useful when we are programming graphics using `\pdfliteral`.

## More programming tools

The list of macros provided for macro programmers cannot be complete in this short article. There are many macros specialized for particular problems like math macros, color macros, font macros, reference macros, citation–bib macros, etc. See the OpTEX documentation [3] for more information. There are plenty of macro programming tips on the OpTEX tricks page [4] too. Here, I will demonstrate only two more cases of useful macros.

**First case:** `\replstring\buff{`⟨*from*⟩`}{`⟨*to*⟩`}` replaces all occurrences of ⟨*from*⟩ text by ⟨*to*⟩ text in the "buffer" macro `\buff`. For example:

```
\def\buff{A text is here.}
\replstring\buff{ }{{ }}
\ea\foreach\buff \do{[#1]}
It returns: [A][ ][t][e][x][t][ ]%
            [i][s][ ][h][e][r][e][.]
```

We have used `\replstring` in this example to "protect" spaces. Each space is replaced by `{ }`. So, the next macro `\foreach` (which reads token by token via an internal macro taking an undelimited parameter `#1`) can read spaces too.

**Second case:** We can set the current typesetting position anywhere by `\setpos[`⟨*label*⟩`]` and then read this position elsewhere with the (expandable) commands `\posx[`⟨*label*⟩`]`, `\posy[`⟨*label*⟩`]` and `\pospg[`⟨*label*⟩`]`. The first two commands return the $x, y$ coordinates of the absolute position of the `\setpos` point on the page (measured from the left-bottom corner). The values are given in the format ⟨*number*⟩`sp`. You can convert to `bp` units (for example) with `\bp{\posx[`⟨*label*⟩`]}` or read into a variable with `\mydimen=\posx[`⟨*label*⟩`]\relax`. The last one (`\pospg`) returns the global page number of the document where the `\setpos` point was set. The data is available after a second run of TEX because an external `.ref` file is used for this purpose.

## Writing public macro packages

When you are writing macros for your usage, there are no rules for naming the control sequences. You can write any macro code and test it. If you plan to release such a macro code as a public package, however, then I recommend the following naming conventions described here. You can look at the code of the `math.opm` package [5] for inspirations and examples of how to create packages for OpTEX. This package deals with options, math macros, and there is a special section about writing public packages too.

First of all, you may set a package shortcut. I'll use the shortcut `pkg` in the following examples. If you select a shortcut used by another package, then users are unable to load both these packages at one time: OpTEX reports an error. So, it is a good idea to see what public packages for OpTEX are available and thus choose a shortcut that isn't already being used.

First, two code lines (after optional comments) in the package file (which should be named `pkg.opm` for our example) should be

```
\_def\_pkg_version {0.07, 2023-01-14}
\_codedecl \supermacro {Title <\_pkg_version>}
```

The first argument of the `\_codedecl` macro (in this example, `\supermacro`) is a macro name that `\_codedecl` checks for being already defined; if it is, `\endinput` is executed, so that the package is not read twice. The idea is that `\supermacro` is a macro never used before and will be defined in this package. The second argument of the `\_codedecl` macro is printed to the log file. The `Title` should be a short title for the package.

The macro code that follows has to be surrounded by

```
\_namespace{⟨package-shortcut⟩}
...
\_endnamespace
```

I. e. `\_namespace{pkg}`...`\_endnamespace` in our example. Also, the `\_endcode` macro has to be called just after `\_endnamespace`. It is similar to `\endinput`, but has more features (described below).

Suppose that you have tested your macros with names in the public namespace. Now, rename all used control sequences by the following rules:

- If it is a TEX primitive or an OpTEX macro, add the "`_`" prefix: use `\_foo` instead of `\foo`.
- If it is your macro, defined and used in the package, add the "`.`" prefix.

Each `\.foo` is transformed to `\_pkg_foo` automatically inside the `\_namespace`...`\_endnamespace` scope. A macro programmer is thus not forced to write and read his package shortcut again and again for essentially all internal control sequences in the macro code.

If you decide that a macro is intended for users in the public namespace, export it from the package namespace to the public namespace using:
`\_nspublic`⟨*list of control sequences*⟩`;`
In our example, we could do:

```
\_def \.supermacro #1#2#3{...}
\_nspublic \supermacro ;
```

The `\_pkg_supermacro` and `\supermacro` control sequences are now defined, with the same meaning.

The `\_nspublic` command checks if the given macro is defined already in the public namespace. If so, then it is redefined, but a warning about it is shown on the terminal.

Maybe there is no reason to declare both the internal copy of a control sequence `\.foo` and the public copy `\foo`. You can declare `\foo` directly, as in `\_mathchardef\foo`, `\_newcount\foo`, `\_def\foo`, etc. But it is highly recommended to prefix such a declaration by `\_newpublic`. For example:

```
\_newpublic \_newcount \foo
\_newpublic \_mathchardef \bar = "123456
```

This prefix does the same check as `\_nspublic`: if a declared control sequence is already defined, it is redefined but with a warning printed.

You can add documentation text to individual macros in a `\_doc`...`\_cod` block. These parts are skipped when your macros are read. For example:

```
   \_doc
   The \`\supermacro` reads parameters and does
   a supertrick A and then does a supertrick B.
   \_cod

\_def \.supermacro #1#3#3{...A...B.}
\_nspublic \supermacro ;
```

It is recommended to append more extensive documentation of the package after the `\_endcode` command. This text is not read, because `\_endcode` executes `\endinput`. This way, you have code and documentation together in a single file, making it much more convenient to manage the package.

You can append a special block `\_doc`...`\_cod` to the documentation after `\_endcode`, to include commands used by the `\docgen` command from OpTEX. Typical usage of this final `\_doc`...`\_cod` scope could be:

```
\_doc
   \load [doc] % provides \printdoc, etc.

   \tit Package which enables super-\TeX/ing
   \hfill Version: \_pkg_version \par
   \centerline{\it Au. Thor\/\fnotemark1, 2023}
   \fnotetext{\url{https://au.thor.or}}

   \notoc\nonum\sec Table of contents
   \maketoc

   \printdoctail pkg.opm % prints the doc.
                 % written after \_endcode
   \sec Implementation
   \printdoc     pkg.opm % prints doc. of code
                 % before \_endcode

   \nonum\sec Index
   \begmulti 3
      \tt \makeindex % prints index, 3 columns
   \endmulti
   \bye
\_cod
```

The macros provided by `\load[doc]` are described in the OpTEX documentation [3], section 2.40.

Now, you have everything you need in a single file: the code itself, technical short documentation, detailed documentation, and commands to generate a whole document including title, table of contents, index, etc. The macro code is ready to be used directly without docstrip pre-processing.

A user can load your package with `\load[pkg]` or can generate a complete documentation by the command line:

```
optex -jobname pkg \\docgen pkg
```

You can try to use this command for the real existing package:

```
optex -jobname math \\docgen math
```

Run this command three times because TEX needs to generate the correct table of contents and the index.

If you write a package for OpTEX, please let me know about it. I'll add a notice about it into [3], section 1.7.3.

## References

1. OpTEX. petr.olsak.net/optex/
2. P. Olšák: TEX in a Nutshell. 2020, 30 pp. https://petr.olsak.net/ftp/olsak/optex/tex-nutshell.pdf
3. OpTEX manual. https://petr.olsak.net/ftp/olsak/optex/optex-doc.pdf
4. OpTEX tricks. https://petr.olsak.net/optex/optex-tricks.html
5. OpTEX macros for doing math more comfortably. https://petr.olsak.net/ftp/olsak/optex/math-doc.pdf

◇ Petr Olšák
Czech Technical University
in Prague
https://petr.olsak.net

# Reflections on `\globaldefs` in plain TeX

Udo Wermuth

**Abstract**

This article discusses a single integer parameter of the program TeX: It looks at the history of the invention of `\globaldefs`, describes the behavior of this internal parameter and tries to list useful applications. It also warns about constructions that might lead to faulty results. Moreover, it explains why one must do a careful verification of someone else's code if that code should be reused under a setting of `\globaldefs`.

## 1 Introduction

The core design model of TeX, the box/glue/penalty model, hasn't changed since its creation in April 1977 [9, p. 142]. And one of the model's realizations hasn't changed much either: After the debugging in March 1978, the code of the line-breaking algorithm was only once overhauled (June 1980) [9, ch. 3] and received in November 1982 an important fix in one of its computations [7, pp. 274–276]. But nevertheless the program TeX mainly grew because of the addition of new *internal parameters* and *primitives*, i.e., commands that are implemented in the program and aren't defined as macros. Luckily, we are able to follow how TeX evolved as TeX's author, Donald E. Knuth, decided to keep a detailed log file about all changes that he applied since March 10, 1978; see [5]. This log contains the descriptions of bug fixes as well as enhancements like *generalizations*. The above mentioned changes to the line-breaking algorithm are the entries #461 and #554.

Knuth stated that he wanted to create a typesetting language but early users pushed him to add more and more programming features [9, p. 648]. Obviously, his intention was to keep the language TeX concentrated on its application domain: typesetting texts.

**A new parameter.** One generalization is listed in the aforementioned log as entry #623, dated January 20, 1983: "Add a new `\globaldefs` feature." Knuth describes that a generalization often occurred when people presented him with new applications [7, p. 281]: "When I couldn't handle the new problem nicely with the existing TeX, I usually would end up changing the system." From other notes in the log file one sees that he was working on *The TeXbook* chapters 16 and 17 and on Appendix G; these are the chapters about typesetting math. The parameter `\globaldefs` has neither a direct influence on the typesetting of mathematics nor is it used in the text of *The TeXbook*. Moreover, the entry in the log doesn't carry a name of another person as is done with other entries (see, for example, entry #631). It seems that no one individual suggested the creation of this parameter.

I guess that the idea for this generalization was born during the work on entry #621, which is labeled "a cleanup for consistency or clarity". In [7, p. 245] this type of enhancement is described as: "Here I changed the rules of the language to make things easier to remember and/or more logical." The cleanup of January 19 touches TeX's procedure *prefixed_command* [4, §1211] which got, with #623, the code for the parameter `\globaldefs` in §1214.

Thus, I surmise the question "Why was the parameter `\globaldefs` added to TeX?" should receive an answer like "because this generalization was an ad hoc idea about a programming feature that could easily be implemented" instead of "because a detailed study showed that this programming feature lets TeX gain a lot of capabilities to solve certain problems more nicely."

Please note, I use "ad hoc" without implying that the implementation was spontaneous or unplanned. Knuth defined the parameter `\globaldefs` one day after #621. Thus he had time to think about it and to design its implementation.

The creation of `\globaldefs`, as documented in entry #623 of [5], didn't implement the functionality that this feature has today. On June 7, 1983, Knuth states in entry #710 that he made a cleanup for the rules of `\globaldefs`. One month later he had to apply a fix to this feature; see entry #748. It's a fix of type "forgotten function" which is described as [7, p. 245]: "Here I didn't remember to do everything I had intended, when I actually got around to writing a particular part of the code." Section 2 explains the impact of these two changes.

**Not much coverage in the literature.** *The TeXbook* [3] mentions `\globaldefs` in only three places. No exercise applies this parameter nor is it used in the format file `plain.tex`. A brief description appears between two blocks with syntax rules on page 275. Pages 206 and 215 indirectly explain fix #710; the impact of #748 is mentioned on page 238.

Well, *The TeXbook* cannot explain everything in detail but other books about TeX might fill the gap. Unfortunately, all books that I own either don't mention `\globaldefs` or they just repeat what is written in *The TeXbook* on page 275. At last I found on CTAN one that uses it: [1, p. 306] applied `\globaldefs` mainly to save keystrokes.

doi.org/10.47397/tb/44-1/tb136wermuth-globaldefs

So what are the use cases of `\globaldefs`? Under which conditions does the saving of keystrokes create a *real* benefit? And is it only used to save keystrokes or are there other useful applications?

Another place to search for use cases is the archive of *TUGboat* [10]. I found only one article by another author; the text describes how to combine TEX-formatted labels with PostScript files; see *TUGboat* **13**:3, page 332. Nowadays it is difficult to say why the parameter is used as the code to which `\globaldefs` is applied isn't completely known. But it's very likely that its use merely avoids the input of a handful of `\global`s. Again, `\globaldefs` is used to save keystrokes.

One can find `\globaldefs` in code on CTAN too. As an example, look at `twimac.tex` by Knuth [6]. It's a macro package to support the program TWILL [8] and contains `\globaldefs` four times. He used this program in 1985 to create the extremely useful mini-indexes for *TEX: The Program* [4].

**Contents.** The rest of this section briefly reviews the concepts known as "local" and "global". Section 2 describes what `\globaldefs` does including all special cases. Section 3 looks at the applications of a non-default setting for `\globaldefs`; it lists four use cases. Another non-default setting is analyzed in section 4. At its end I formulate a fifth use case with a *recommendation* for the use of `\globaldefs`. Section 5 discusses what can happen if `\globaldefs` with different settings are nested. At the end of this section I extend the recommendation of section 4. Section 6 briefly examines technical aspects inside of TEX, especially a possibility to save memory. The article ends with some personal remarks in section 7.

**Local and global.** It is well known that TEX obeys with assignments and macro definitions the group level in which it executes the command.

**Example 1: Local assignment restored**

```
\dimen0=10.01pt
{\dimen0=20.02pt A: \the\dimen0 }% a group
\quad B: \the\dimen0
```

**TEX output**

A: 20.02pt B: 10.01pt ▯

(The rectangle in the gutter — here at the end of the part "TEX output" — marks the end of the example.) TEX prints `20.02pt` and then `10.01pt` because the second assignment occurs inside a group. The new value is only *locally* known, i.e., only inside this group level. One must apply the prefix `\global` to change `\dimen0` inside the group with a *global* effect, i.e., to keep the new value after the '`}`'. Then the code prints `20.02pt` twice.

**Example 2: Global assignment kept**

```
\dimen0=10.01pt
{\global\dimen0=20.02pt A: \the\dimen0 }%
\quad B: \the\dimen0
```

**TEX output**

A: 20.02pt B: 20.02pt ▯

But, let's note that not all statements that appear as if they were an assignment are such to TEX. For example, `\openout` associates a stream number to a file name and accepts an equal sign in its syntax but isn't an assignment. It cannot be prefixed by `\global`. Well, it doesn't need to be prefixed as the association is by default global.

There are a few commands and quantities which TEX treats with global effect. For example, changing a box dimension, i.e., an assignment to `\wd`, `\ht`, or `\dp` of a box, is always global in the sense that the box dimension is changed for this box permanently. Only if the box number is restored at the end of a group the restored box has its former dimensions.

Some primitives that always act globally are simple commands without the form of an assignment. For example, a switch of the interaction mode from the default `\errorstopmode` to `\batchmode` is a global change. See page 277 of *The TEXbook* [3] for the syntactic rule ⟨global assignment⟩ that collects all the statements that act globally. One can use the prefix `\global` in front of `\errorstopmode` or `\batchmode` but it doesn't make a difference.

## 2 What does the parameter do?

The value of the integer parameter is simply changed by an assignment of the form `\globaldefs` $= n$, in which $n$ is any valid TEX integer. But the parameter `\globaldefs` acts only in three different ways because TEX only checks if $n > 0$, $n = 0$, or $n < 0$.

- With $n > 0$ TEX starts to execute commands listed under ⟨simple assignment⟩ (see [3, p. 276]) and ⟨macro assignment⟩ (see p. 275) — in short: all assignments, arithmetic commands, and all control sequence definitions including `\font` and `\read` — as if the prefix `\global` was specified. Thus, TEX doesn't care if `\global` occurs in the code or not. It operates on these statements as if it saw the prefix.

Note that TEX must execute the code to assign the prefix; for example, a `\def` inside a `\def` doesn't become global during the definition. Similarly, the `\relax` equivalent of a `\csname`/`\endcsname` construct in a test or after `\expandafter\show`, etc., which is otherwise an undefined control sequence, does not become a global `\relax` as it isn't executed.

- With $n = 0$ `\globaldefs` is neutral or switched off; this is the default value of the TEX program.

Thus, a statement that's influenced by `\global` (except those that always act globally) has to receive the prefix `\global` to change a value not only inside the current group level; see example 2.

• With $n < 0$ `\globaldefs` switches the prefix `\global` off: `\global` has no impact on the following command and operates like `\relax`. But as a prefix it must still be used only with ⟨simple assignment⟩ and ⟨macro assignment⟩. For example, you cannot write `\global\begingroup` as `\begingroup` doesn't accept `\global` and therefore TeX raises an error.

The primitives `\gdef` and `\xdef` behave like `\def` and `\edef`, respectively; this was the cleanup in #710 of [5]. Thus, we can treat `\gdef` and `\xdef` as equivalent to `\global\def` and `\global\edef`.

The always-global commands listed as ⟨global assignment⟩ in [3] keep their global effect.

**Get `\globaldefs`' value.** As an internal parameter `\globaldefs`' value can be shown, printed, or assigned to an integer register or parameter. That is, `\showthe\globaldefs` shows the current value on the terminal; `\number\globaldefs` typesets its value (use it in math mode as the value can be negative), and, for example, `\count9=\globaldefs` stores its value in the count register number 9.

**A special case.** When TeX scans the tokens in the preamble of an `\halign` or `\valign` it collects them for the templates of the rows or columns, respectively. TeX processes a few tokens during this scan: (1) the alignment character, '`&`'; (2) the expand token, `\span`; (3) the parameter character, '`#`'; (4) the end-of-preamble tokens `\cr` and `\crcr`; and (5) the `\tabskip` token with its following glue specification. Everything else belongs to the templates.

This means that `\global` cannot be applied to assignments to `\tabskip` in the preamble as TeX puts this `\global` into the template. It also means that the `\tabskip` cannot be placed in a group to make the assignment local: Even in such a case the new `\tabskip` value in the preamble is extracted and used for the space inserted between the following columns or rows. But at the end of the alignment TeX forgets all non-global changes to `\tabskip`. The last change to `\tabskip` in the preamble never determines its value after the alignment except when the alignment starts under `\globaldefs` $> 0$. Then all assignments in the alignment are global; with fix #748 of [5], including the ones to `\tabskip` in the preamble, as probably anticipated by the users.

## 3 Use cases for `\globaldefs=1`

Two use cases for `\globaldefs=1` were already mentioned. In section 1 we learned that it can be used to avoid the repetitive input of `\global`. Section 2, subsection "A special case", showed that it's required to make a `\tabskip` in a preamble global.

**UC1: global `\tabskip`.** As explained in section 2 a `\tabskip` assignment in the preamble cannot use `\global`. Inside the table entries the value can be globally modified: use `\global\tabskip` followed by a glue specification. It has no effect on the rôle of `\tabskip` in the preamble: The white space between columns or rows of the alignment isn't changed.

To be honest, I hesitate to call this a use case for `\globaldefs=1` as it is quite extreme to apply this setting to have global assignments to `\tabskip` in the preamble. Without `\globaldefs=1` just insert `\noalign{\global\tabskip=\tabskip}` after the preamble's `\cr` to make the preamble's last value of `\tabskip` global; no other statement is affected.

**UC2: saving keystrokes.** Sure, a `\globaldefs=1` followed by at least four statements that should otherwise be prefixed by `\global` and a `\globaldefs=0` can save at least 2 keystrokes as the prefixes aren't required in the input: $2 \times 13$ keystrokes vs. $4 \times 7$. Let's write it explicitly although I use just two assignments not four; you see later why.

**Example 3: Saving keystrokes with `\globaldefs=1`**

```
\globaldefs=1 \count9=123 \dimen9=123.45pt
\globaldefs=0
```
□

One should observe that the second assignment to `\globaldefs` is a global assignment too. But this assignment can be avoided as all statements are global except the `\globaldefs=1`; so, inside a group, TeX restores only the value of `\globaldefs`. Now one saves a keystroke if there are just two assignments.

**Example 3 continued: Simpler input**

```
{\globaldefs=1 \count9=123 \dimen9=123.45pt }
```
□

One shouldn't do this without need. (If you have to create the group you don't save keystrokes.) Check [1] and the first occurrence in [6] and you see that the authors are *forced* to open a group because of a catcode change; it might be hidden in an `\obeylines`. The setting `\globaldefs=1` allows to enter the code that belongs to the outer level of the macro package as if it were not in a group. Often in such a group, only `\def` is used; thus, one saves only a '`g`' and not a '`\global`' in the statements.

But keystroke savings aren't the only point. If the catcode change isn't needed anymore one can easily remove the group and the `\globaldefs=1`. Then the code integrates well with the rest. (See, for example, file `ctwimac.tex` in the directory of `twimac.tex`; the first `\globaldefs=1` of the latter isn't used in the former anymore.)

The application of \globaldefs=1 in front of a loop is similar. Here is an example using the prefix \global (such a case appears in [6]):

**Example 4: Several \global in a loop**

```
\newcount\nn \newcount\maxnn \maxnn=200
\global\nn=100
\loop \global\count\nn=0 \global\dimen\nn=0pt
   \global\skip\nn=0pt \global\muskip\nn=0mu
   \global\toks\nn={}%
\ifnum\nn<\maxnn \global\advance\nn by 1 \repeat
```

To limit the scope of \globaldefs=1 a group encloses the whole \loop/\repeat construction.

**Example 4 continued: Loop with \globaldefs=1**

```
\newcount\nn \newcount\maxnn \maxnn=200
{\globaldefs=1 \nn=100
 \loop \count\nn=0 \dimen\nn=0pt \skip\nn=0pt
    \muskip\nn=0mu \toks\nn={}%
 \ifnum\nn<\maxnn \advance\nn by 1 \repeat}
```

One advantage is that the material between \loop and \repeat is more compact and might be easier comprehended. A side effect in this scenario is that the \body in the macro \loop becomes global too.

**UC3: global expand.** Here \globaldefs=1 is applied to a token register, a macro, or a TEX file that contains definitions and assignments. When TEX expands the register with \the, executes the macro, or inputs the file all statements receive \global. Such a construction can be used, for example, inside the output routine to make the data in the register or the macro available to the outer level. (For a better but more advanced example, read "Processing by TEX" in [8, p. 7] together with [6], if you can understand high level descriptions of output routines.)

**Example 5: Apply \global to a collection**

Here is a very simple example of how statements that were collected in a token register are executed inside a group with \globaldefs=1. Assignments to \hsize and \vsize via \setsize and macros to get their product are placed into the register. I omit all error checking.

**TEX input**

```
{\catcode`\_=11 \newcount\area_sqmm
 \newtoks\area_cmds % the collection
  \global\area_cmds={\area_sqmm=0 }% initialize
 \gdef\set_area(#1){% #1: dimen w/o unit mm
  \ifnum\area_sqmm=0 \area_sqmm=#1\relax
  \else \multiply\area_sqmm by #1\fi}
 \gdef\setsize#1#2mm{% #1: h/v; #2: dimen w/o mm
  \area_cmds=\expandafter{\the\area_cmds
   \csname#1size\endcsname=#2mm\set_area(#2)}}
 \gdef\prtarea{\message{Area: \the\area_sqmm
   sqmm.}\area_cmds={\area_sqmm=0 }}% reset
}\setsize h176mm\setsize v250mm% fill collection
{\catcode`\_=11 \globaldefs=1 \the\area_cmds}
\prtarea % shows 176mmx250mm = 44000sqmm
```

Without the \globaldefs the \hsize and \vsize settings aren't global and \area_sqmm would be zero in the message. We can add \global to the definitions, but then the collection cannot be executed without \globaldefs=-1 for a local application.

**UC4: keep code and output in sync.** Journals like *TUGboat* publish TEX input and also its typeset output. To stay in sync *TUGboat* suggests to use the following construction; here demonstrated with additional code from me. The first example of this article was more or less coded as

```
\verbatim[\inputfromfile{example1.tex}]
\endverbatim\exout \input example1.tex \exend
```

where the macro \exout outputs "TEX output" in boldface and opens a group that ends in \exend. The first line reads example1.tex and typesets its contents verbatim. Line 2 executes the code in this file inside the \exout/\exend group. In this group and in front of the \input a register value from a previous example might be entered or a parameter implied from the current topic like \parfillskip is set. Example 1 doesn't need anything of this kind.

Here is an example in which the environment of \exout might start with \globaldefs=1. In this example a file is opened for writing. Just one line is written to this file and this line contains a macro that was defined inside the example. TEX executes the \write delayed, i.e., the file gets the line with the next \shipout; see [3, pp. 226–227].

**Example 6: Code and its output**

First, we look at the contents of the file example6.tex.

**TEX input**

```
\toks9={Hello world!}\openout5=ex6outfile.tex
\def\textforex6{\number\pageno: \the\toks9 }%
\write5{\textforex6}\closeout5
```

This code cannot be executed inside a group. The \write waits for the next page break and when it occurs TEX expands the token list of this \write and stumbles over \textforex because outside of the group it's undefined. One could add \immediate to \openout, \write, and \closeout; or use \gdef for \textforex and \global in front of the \toks assignment. This destroys the example if the author wants to keep the code as simple as possible.

We don't want to change anything in the input file example6.tex that contains the code of the example. And we want to keep the code of \exout (and \exend). Thus we must make the assignment and the definition global, i.e., we must use \globaldefs.

**Example 6 continued: Code used for execution**

```
\exout\globaldefs=1 \input example6.tex \exend
```

We are only allowed to do this as we know the code in the file. It doesn't work always; see example 11.

## 4 Use cases for `\globaldefs=-1`

Of course, the use of `\globaldefs=-1` can be easily replaced if the code to which it should be applied doesn't contain other settings of `\globaldefs`. As `\relax` passes prefixes on to the next token we can code `\let\global=\relax` and the code `\long \global\def` still generates a `\long\def`.

But there's a difference between an deactivated `\global` by `\globaldefs=-1` and the above `\let`. If you scan tokens one by one and compare them against `\relax` then the new `\global` executes a wrong branch of the test. The solution consists of a replacement text that uniquely identifies `\global` as well as `\gdef` and `\xdef`.

**Example 7: Avoiding `\globaldefs=-1`**

```
\let\CPglobal=\global \let\CPgdef=\gdef
   \let\CPxdef=\xdef
% use for them unique replacement texts
\def\global{\relax\relax}\def\gdef{\relax\def}%
   \def\xdef{\relax\edef}%
... % code with inactive \global, \gdef, \xdef
% restore \global, \gdef, \xdef with the copies
\let\global=\CPglobal \let\gdef=\CPgdef
   \let\xdef=\CPxdef
```
□

**Use cases from section 3.** Obviously, one cannot save keystrokes with the setting `\globaldefs=-1` as only existing `\global` tokens are affected. So there are no use cases that correspond to UC1 or UC2. UC3 can be turned into a "local expand" variant if the collection contains the prefix `\global`. In the scenario of UC4 `\globaldefs=-1` can only be used if the code doesn't contain *necessary* `\global`. Thus, it helps in some sense only for badly written code.

**Example 8: Case where `\global` is necessary**

Is `testscript.sh`'s first line a so-called *shebang line*, signaling that it is a *Bourne shell script*?

**TEX input**

```
\def\uncatcodespecials{% see The TeXbook, p. 380
   \def\do##1{\catcode`##1=12 }\dospecials}
\edef\shebangline{\string#!/bin/sh}%Bourne shell
% the code with \read in a group
\newread\infile \openin\infile=testscript.sh
\def\readin{{\uncatcodespecials \endlinechar=-1
 \global\read\infile to \lineofinfile}}\readin
\ifx\lineofinfile\shebangline
   \message{Bourne shell}\fi \closein\infile
```
□

If the `\global\read` isn't global because of an active `\globaldefs=-1` then the `\ifx` doesn't produce a reliable result as `\lineofinfile` is either undefined or contains data from another assignment.

**Reuse unknown code.** One might think a good use case for the parameter `\globaldefs` with a negative number is to limit the effect of a file that contains macros. But of course, `\globaldefs=-1` must not eliminate a necessary `\global`. One must be careful if it should be applied to reuse code.

For example, assume that `calmacros.tex` contains macros for calendrical computations like the Day of Repentance and Prayer for a given year. (It's celebrated eleven days before the first Sunday of Advent, so its month is November.) Another file consists of macros that belong to the same domain; it's very likely that the files share, for example, register names. Assume that in the second file the Day of Repentance and Prayer is required and that the package `calmacros.tex` provides a macro with the name `\CalcRepPrayDay` to compute that day.

**Example 9: Avoiding global changes with `\input`**

```
\newcount\repprayday {\globaldefs=-1
\input calmacros \repprayday=\CalcRepPrayDay2023
\globaldefs=0 \global\repprayday=\repprayday}
```
□

The main file has then access to `\repprayday` but all macro names, register names etc. of `calmacros.tex` are gone when the group ends. Nevertheless, example 8 warns us: the result might be wrong!

Even if the code doesn't throw an error one cannot trust the result. In `calmacros.tex` computations might occur inside a group and the final result made available to the outside only through an assignment prefixed by `\global`. This doesn't happen if `\globaldefs=-1`. Thus, the result value is restored when the group ends; the result becomes a "random" value.

One must verify that a file with macros that one wants to reuse in a group with `\globaldefs=-1` contains at most unnecessary `\global`s in the code paths that are called.

Another more concrete example from this text:

**Example 10: Which number is output after "A:"?**

```
\globaldefs=-1 {\input example3 }%
A: $\number\globaldefs$
```
□

The result is either 0 or $-1$; it depends if the black box `example3.tex` refers to the version with the two `\globaldefs` or to the one with the group. Moreover, we also cannot answer the question if we use `\globaldefs=1` instead of `\globaldefs=-1`.

What has been found out is known [2] but it should become common knowledge.

**UC5: reuse known code.** To eliminate the effect of `\global` one might execute code inside a group with the setting `\globaldefs=-1`. But one must verify that it doesn't deactivate a necessary `\global`.

In general follow this recommendation: Never apply `\globaldefs` $\neq 0$ to code that you don't completely know or fully understand in all its details because you might get a random result.

## 5 Nested `\globaldefs`

A programmer might ask if code can be written in such a way that it protects itself against bad results if someone reuses this code inside a group with a non-zero `\globaldefs`. Of course, there is a simple solution: Start your code with `\globaldefs=0`. This cancels a `\globaldefs=-1` and with `\globaldefs=1` only the `\globaldefs=0` becomes global.

Let's assume that we want to execute all statements under the setting of `\globaldefs` except if this would cause an error. Then the protection with `\globaldefs=0` is a valid solution for `\globaldefs=-1` if the reused code has only necessary `\global`s. For `\globaldefs=1` it isn't a solution.

**Protection against `\globaldefs=1`.** Let's state the goal precisely. A programmer wants to protect code so that it still executes correctly if someone takes this code and places it inside a group starting with `{\globaldefs=1`. To simplify the discussion `}\globaldefs=0 \global\globaldefs=0` is used at the end of the group; thus a change of `\globaldefs`' value inside the group isn't important. The solution to start the code with `\globaldefs=0` is not considered to be valid. Only the code that *must be protected* because otherwise the original code does something wrong *should be protected*. Everything else should be executed using `\globaldefs=1`.

I admit it sounds like an unrealistic scenario. But we might learn from it.

**Example 11: Try to protect against `\globaldefs=1`**
The following code should be protected to allow execution in a group that sets `\globaldefs=1`. It's artificial code to keep the size of the often repeated example small.

**TEX input**

```
\dimen9=1000pt \count9=0
{{\catcode`\e=3 ex^2e}%
 \global\advance\count9 by 2 }%
\divide\dimen9 by\count9 \count8=\count9
```
□

It's clear why this code cannot be executed under a `\globaldefs=1` without throwing an error. The catcode change becomes globally active and thus an undefined control sequence `\advanc` is reported later.

A setting `\globaldefs=-1` or `0` must be placed in front of the catcode change to keep it local in its group. Let's apply −1.

**Example 11 continued: A failed attempt**

```
\dimen9=1000pt \count9=0
{{\globaldefs=-1 \catcode`\e=3 ex^2e}%
 \global\advance\count9 by 2 }%
\divide\dimen9 by\count9 \count8=\count9
```
□

This code throws an error too. As TEX executes the new assignment globally, `\globaldefs=-1` survives the end of the inner group and deactivates the `\global` in front of the `\advance`. Thus, TEX restores `\count9` at the next `}`, finds a division by 0 in the last line, and reports "Arithmetic overflow".

Thus, the code for the protection needs an improvement. The `\global\advance` must stay global. We can put a `\globaldefs=0` in front of that statement. But as another group ends, the next line isn't globally executed. Should we use `\globaldefs=0` instead of `\globaldefs=-1` in the inner group?

**Example 11 continued: A successful attempt?**

```
\dimen9=1000pt \count9=0
{{\globaldefs=0 \catcode`\e=3 ex^2e}%
 \global\advance\count9 by 2 }%
\divide\dimen9 by\count9 \count8=\count9
```
□

This code runs without generating an error and it protects the code. But the last statements are still protected although that shouldn't happen. In essence it's more or less equivalent to the initial solution which was rejected above because of this effect.

We must find another solution: Let's keep the `\globaldefs=-1` in the inner group to signal that the catcode change must be local; but its influence must be stopped when this group ends. As the code must work with initial values 0 or 1 for `\globaldefs` its value should be reset after the other group.

**Example 11 continued: The final attempt**

```
\dimen9=1000pt \count9=0
\edef\SAVEglobaldefs{\number\globaldefs}%
{{\globaldefs=-1 \catcode`\e=3 ex^2e}%
 \globaldefs=0 \global\advance\count9 by 2 }%
\globaldefs=\SAVEglobaldefs
\divide\dimen9 by\count9 \count8=\count9
```
□

The current value is captured in a macro and then restored at the correct place. This does what was requested above. But, in this version, two of the original four lines contain additional code and the other two lines are now accompanied by new lines. The amount of code is nearly doubled.

**Extension of the recommendation.** I don't suggest that programmers protect their code against an execution with a non-zero setting of `\globaldefs`; at least not with a `\globaldefs=0` at the start of the file. But we should extend the recommendation stated in UC5: You are responsible to protect the code that you reuse from generating erroneous output because of your setting of `\globaldefs`. And you are responsible to ensure that `\globaldefs`' value outside of the group that you opened is restored if that is required.

## 6 Technical advantages

Up to now we looked at the parameter `\globaldefs` to see what advantages its application has in the input of a user. But, of course, there are technical payoffs too. The first is obvious: An input file needs fewer bytes, i.e., it needs less storage space and might load faster, if at least two `\global` are saved for each `\globaldefs=1`. Is there anything more about the introduction of `\globaldefs`? Can it save memory? Knuth worked hard in the late 1970s and early 1980s to get TEX into the then-available memory space of the then-available computers.

Let's do a little experiment with the two versions of example 4: Execute the two code snippets with `\tracingstats=1` in front of the code and with an `\end` after it. Next, compare the statistics at the end of the `log` files. The `\globaldefs` variant saves twelve memory words on my system.

The effect seen in example 4 doesn't occur always. For other scenarios the number of memory words doesn't change. For example, create two files. Once "`\tracingstats=1 \global\count3=4 \end`" and the second replaces the global assignment by "`\globaldefs=1 \count3=4`". Except for the value of *buffer size* the statistics are identical. The result is the same if `\count` is replaced by `\dimen`, `\skip`, `\muskip`, or `\toks` with appropriate right hand sides. Even a block with all five register types has the same number of memory words.

The opportunities to apply `\globaldefs` are quite rare. Thus, we cannot hope that it helps to save any significant amount of memory in a project.

## 7 Personal remarks

I confess that I haven't used `\globaldefs` often in my TEX projects. I used it when I was forced to do so in an unusual macro project (*TUGboat* **43**:1, p. 63) to protect the code from the problems of section 5. (I suggest on p. 72 to use `\let\globaldefs=\undefined` as the protection cannot be perfect.) There are other TEX primitives that I seldom use, for example, `\valign`; but I thought I had used this primitive more often. It is part of the typographic language that Knuth wants to put into the foreground. I assumed Knuth had a good reason to add `\globaldefs` to the program TEX. At least its addition makes the language more complicated to learn.

I don't deny that UC2 is useful: Saving keystrokes is a nice feature and example 4 looks much better with `\globaldefs=1`. But its use could be easily avoided here as well as in UC3. Only UC4 needs `\globaldefs` if code shouldn't be changed. But an alternative with changed code isn't hard to create: Use `sed` (or similar) to insert `\global` automatically into the code and write a new file that's used after `\exout`. I'm convinced that UC5 is of limited use. And I would never apply it to `\newread` as in `thumbpdf.sty` on CTAN; it isn't my programming style to use such tricks. So I asked: Has a TEX without `\globaldefs` problems that must be solved with this parameter? Is it important to save keystrokes?

The reader might say, "Wasn't the introduction of `\xdef` in #370 of [5] similar, as it just saves one `\global`?" No, I think it does more: The language becomes easier to learn with the pairs `\def`/`\gdef` and `\edef`/`\xdef`. And Knuth was asked to implement this change.

I wrote this article to understand `\globaldefs` better. I learned: All listed use cases have other solutions without complicated tricks. Moreover, nesting `\globaldefs` can create problems as described in section 5. And even with the results of section 6, I wonder why `\globaldefs` became a part of TEX.

## References

[1] Paul W. Abrahams, Kathryn A. Hargreaves, Karl Berry, *TEX for the Impatient*, 2003. `ctan.org/tex-archive/info/impatient/book.pdf`

[2] David Carlisle, comment on `tex.sx`, 2022-06-30. `tex.stackexchange.com/questions/649425/is-it-safe-to-use-globaldefs-for-setting-global-pgf-key-value-pairs/649437#comment1618526_649425`

[3] Donald E. Knuth, *The TEXbook*, Volume A of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1984.

[4] Donald E. Knuth, *TEX: The Program*, Volume B of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1986.

[5] Donald E. Knuth, "The Errors of TEX", *Software — Practice and Experience* **19** (1989), 607–685; reprinted as Chapters 10 and 11 in [7], 243–339. The log, i.e., Chapter 11, is still updated: `ctan.org/tex-archive/systems/knuth/dist/errata/errorlog.tex`

[6] Donald E. Knuth, `twimac.tex`. `ctan.org/systems/knuth/local/lib/twimac.tex`

[7] Donald E. Knuth, *Literate Programming*, Stanford, California: Center for the Study of Language and Information, CSLI Lecture Notes No. 27, 1992.

[8] Donald E. Knuth, "Mini-Indexes for Literate Programs", *Software — Concepts and Tools* **15** (1994), 2–11; reprinted as Chapter 11 in [9], 225–245.

[9] Donald E. Knuth, *Digital Typography*, Stanford, California: Center for the Study of Language and Information, CSLI Lecture Notes No. 78, 1999.

[10] *TUGboat*, archive of all publicly available articles. `tug.org/TUGboat/contents.html`

◇ Udo Wermuth
Dietzenbach, Germany
`u dot wermuth (at) icloud dot com`

## Storing Unicode data in TEX engines

Joseph Wright, LATEX Project Team

### 1 Introduction

Unicode has become established over the past three decades as *the* international standard for representing text in computer systems. By far the most common input encoding in use today is UTF-8, in which Unicode text is represented by a variable number of bytes: between one and four. Unicode deals with *codepoints*: a numerical representation for each character. There are in principle 1 114 112 codepoints available, although not all are currently assigned and some of these are reserved for 'private use' for *ad hoc* requirements.

Each codepoint has many different properties. For example, depending on our application, we might need to know whether a codepoint is a (lower case) letter, how it should be treated at a line break, how its width is treated (for East Asian characters), *etc.* Unicode provides a range of data files which tabulate this information. These files are human-readable and are, in the main, purely ASCII text: they are therefore not tied to any particular programming language for usage. The full set of files is available from `unicode.org/Public/UCD/latest/ucd/`: the complete current set as a zip is around 6.7 MiB.

There are of course standard libraries for common programming languages such as C which both load this data and provide implementations of the algorithms which use this data: things like changing case, breaking text into lines and so on. However, these are not readily available to us as TEX programmers. Thus, if we want to be able to properly implement Unicode algorithms, we will need to look at how to load the relevant data and store it within TEX in an efficient manner.

Here, I will focus on how the LATEX team is approaching the data storage challenge. I will show how the particular requirements of implementing in TEX mean we need to use a mix of approaches, depending on exactly which data we are looking at. The current implementation for loading this data in expl3 is available at `github.com/latex3/latex3/blob/main/l3kernel/l3unicode.dtx`, and is read as part of the LATEX 2$\varepsilon$ format-building process.

### 2 The data challenge

With over a million codepoints, even on a modern computer, storing values for every codepoint separately is impractical, particularly when we worry about having multiple properties and needing to access any data value with equal ease.

In some cases, we do not need to record properties for every single Unicode codepoint. We will see that for example with grapheme breaking: most codepoints have the same property value here, so we can tackle data storage by looking just at exceptions. However, there are major problems there. First, there are plenty of properties where we *do* need to track information of most if not all codepoints. There are cases where we might look at using ranges of characters, but the downside to this can be that we get uneven speed of access: that's fine if all of our documents are written in ASCII, but not acceptable if we need to cover a range of input scripts in an even manner.

This is not something that applies only to TEX of course, and it's a problem that the notes from the Unicode Consortium themselves address. The recommended approach is to use what is called a *two-stage* table. This is a way of covering all of those 1 114 112 codepoints without needing to store separate values for every single one, and maintaining fast access for all codepoints. We will see both how that works and how to do it in TEX below.

### 3 TEX aspects

For classical TEX engines, we might think we don't need to cover all of Unicode: the engines are only 8-bit anyway. But we know that we can take that 8-bit input and treat it as codepoints: the LATEX inputenc package has done that for over 30 years. So even if we don't need to cover *all* of Unicode, we need to handle a subset, and it's not always easy to make this a clearly limited and non-expanding subset. So even for these engines, we likely need methods to store a full range of data.

The Unicode engines XƎTEX and LuaTEX present a different question. They do have tables for some Unicode data: `\uccode`, `\lccode`, `\catcode` and so on. But whilst we do need to set those up (so that they have the 'right' values for TEX operations), it turns out they don't offer enough flexibility to track everything needed. Also, if we want to be able to use code shared by different engines, we want to use approaches that work with pdfTEX anyway. For the LATEX team, that's the case, of course.

The experienced TEX programmer might at this stage well be worrying about where I'm thinking of putting all of this data. TEX is very limited in the data structures it provides: macros and some registers. The latter are simply too limited in number, even with the $\varepsilon$-TEX extensions. We can store quite a bit in macros, and rely on the hash table to get fast access, but even that isn't going to scale well for the amount of data we might want, at least without some extra tricks. But there is another, perhaps unexpected, data store we can use, one that will give us quite a bit of headroom: font dimensions.

It turns out that we can set almost as many `\fontdimen` values for a font as we want, but we need to know how many to create for any given font. We don't want to do that for real fonts, but we can load the same font at lots of sizes and use each size as, effectively, an integer array. All that's needed is to pick sizes that the user doesn't care about: we do that by starting at 1 sp and working upwards. There is a limit on the *total* number of `\fontdimen` values, but it's in the millions and we won't get close to that. So we do have a fast random access data structure we can use for storing integer values. Now all we need to do is use this idea efficiently.[1]

## 4 Making it numerical

Before we deal with storing all of the Unicode data we need, there's the question of exactly what we will store. Very few Unicode properties are numerical: they are descriptors of behaviour. However, we can turn most of them into something we can represent by a number. The Unicode 'General Category' property is a good demonstration here. There are 31 possible values, for example

| | |
|---|---|
| Cc | Control character |
| Lu | Uppercase letter |
| Nd | Decimal number |

It's trivial to assign a numerical value to each of these; then we can store that integer value and quickly convert to the descriptor as required.

That approach works for most properties, but not, for example, for case-changing data. The case mapping of a codepoint will itself be a codepoint: it could be the same one, or it could be *anywhere in the Unicode range*. We are going to want values that have some chance of repeating, so storing the *absolute* value of the target codepoint isn't going to work. Instead, we will store the *relative* position of the 'output' codepoint. For example, `A` is `"0041` and `a` is codepoint `"0061`. So we will store the lowercase mapping for `A` as `"0041` − `"0061`, *i.e.* −32. The open-ended nature of the values here is going to impose a few extra conditions, as we'll see in a bit.

[1] If we are working in LuaTeX, other data structures are available for storage. In expl3, the same macro-level interface is used for creating integer arrays in all engines, with LuaTeX using a Lua-based storage method. This allows an engine-neutral approach to the problem of storing large amounts of numerical data whilst still taking advantage of the greater flexibility of Lua where available.

## 5 Two-stage tables . . .

The idea of a two-stage table is that it offers fast data access to a large number of values, while at the same time avoiding storing every single entry separately. This works for us here as there are patterns in the data we can exploit. Two-stage tables are recommended by the Unicode Consortium and are used by several languages. A particularly clear explanation, including an implementation for storing general category data written in Python, is available at `strchr.com/multi-stage_tables`.

The two-stage approach is based on arbitrary data blocks (not Unicode's character blocks): we divide the full Unicode range into equal-sized blocks, then deal with each block separately. The size of the block (a power of two) somewhat affects the amount of compression we will see, but anything from 64 to 256 gives similar results; these are typical values.

Dividing the full range into blocks of known size means of course that we know how many blocks there will be. For example, if we assume a block size of $n = 256$, there will be 4352 blocks. That will be the size of the first table we will use, with one entry for each of these blocks: that means it has a predictable size, and can be created before we do any data processing. Each entry in this first table points to a second table, of which we will need several.

The second stage tables contain the data for each block, so have $n$ entries each. What we don't know here before creating the entire data structure is how many of these second stage tables we will need. At the start of building the structure, each block of codepoints will need a separate second stage table. But as we go on, we will find that different blocks can reuse the same second stage table. So, representing the table as a comma-separated list, we might see our first stage table (the property values we need to store) looking something like[2]

```
1, 2, 3, 1, 4, 5, 6, 1, 2, 8, 1,
...
```

That is, the first three values in the first-stage table each point to different blocks in the second stage table, but the fourth value points to the same second stage block as the first, and so on. As we get into the parts of Unicode that have long ranges of codepoints with identical property values, this compression effect becomes significant and the total size of the two stages ends up much smaller than the total number of codepoints.

[2] Here, I am using an index from 1: this is the approach used by expl3 and by Lua. Languages involving direct memory management will use an *offset* starting from 0.

With this all set up, retrieving a value is quite quick. We can find which block a codepoint is in, and the position within a block, with a couple of numerical expressions. So getting a value out of the tables is very fast. That of course is the point: the work is done in the creation stage, so at point of use everything is very quick.

To set this up in TeX, we need to think about exactly how to create those two tables. As I've said, we can predict the size of the first table, so we can make that directly using the `\fontdimen` approach. We can't do that for the second stage as we don't know in advance how many entries we will need. Also, we want to be able to check each block's table against those we've already created. That's better done if the data are stored in macros: a series of comma lists work well. Macros are fine if we don't need to access the values randomly, and during the creation stage that's true. We can then use fast `\ifx` tests to check each block as we finish it: have we seen this block before? Once we've done all the blocks, we can then create the second `\fontdimen` table in one shot. (We could make lots of stage-two tables, but as they are of predictable size, we can store all the information in a single `\fontdimen` array using an offset to get the right block information.)

With over a million codepoints, one might be worried about how long reading every one of them will take. However, in most cases, large parts of the full range are compressed in the input. For example, `UnicodeData.txt` contains details of case mappings and general category. For many east Asian characters, these and other values are identical, so the file simply lists the first and last entries with similar values. So for these, we don't have to work through every codepoint: we just have to work out which second stage table they use, then add the right number of entries to the first stage.

With some carefully-coded for loops, we can read the entirety of `UnicodeData.txt` and save all of the upper- and lowercase data in a couple of seconds. That needs only four `\fontdimen` arrays, and the total number of entries is fewer than half of the number of codepoints that have case data.

## 6 ... or not

As you will have seen, whilst a two-stage table approach is efficient for covering the whole Unicode range, there is a limit to the degree of compression, as the first stage will always have a significant number of entries, even if we need very few second stage tables. At the same time, the approach relies on being able to read the data once, so it's not so good if we want to make *ad hoc* changes. It should come as no surprise, therefore, that dealing with one-off overrides is best done using other methods, for example storing as macros which can then be looked up using TeX's hash table.

The line between using a two-stage table approach and individual hash table entries (or other approaches) is fuzzy: one needs to make a judgement. But broadly, if we are looking at fewer than a couple of thousand codepoints, we are likely to avoid a two-stage approach. For example, storing case folding and titlecasing information is easier using a macro approach: both are essentially tightly focussed variants of standard case changing, and apply only to a relatively small number of codepoints.

Another area where two-stage tables are more tricky to use is where we need to store multiple values. This applies for example to normal form decomposition and to full lower/uppercasing data. We could do that by having combined values in a first stage table, for example 1 to 999 for the first output codepoint and 1000 to 100 000 for the second. But the alternative of using a two-stage approach for the one-to-one data, then a hash approach for one-to-many, works pretty well for us.

Finally, there is a consideration about how we are actually loading the data. The source data file `UnicodeData.txt` is ordered by codepoint, so is ideal for reading line by line and turning the contents into a few two-stage tables covering the different concepts. Here are a few lines from `UnicodeData.txt`:

```
0000;<control>;Cc;0;BN;;;;;N;NULL;;;;
...
0041;LATIN CAPITAL LETTER A;Lu;0;L;;;;;N;;;;0061;
...
0061;LATIN SMALL LETTER A;Ll;0;L;;;;;N;;;0041;;0041
...
10FFFD;<Plane 16 Private Use, Last>;Co;0;L;;;;;N;;;;;
```

Several of the other Unicode data files are ordered for logical access. For example, the grapheme-breaking data file (`GraphemeBreakProperty.txt`) is divided up by breaking class, then within that ordered by codepoint. Some example lines, from three different classes (in the real file, the comments are not on separate lines):

```
0600..0605    ; Prepend
 # Cf   [6] ARABIC NUMBER SIGN..ARABIC NUMBER MARK ABOVE
...
00AD          ; Control
 # Cf       SOFT HYPHEN
...
AC01..AC1B    ; LVT
 # Lo  [27] HANGUL SYLLABLE GAG..HANGUL SYLLABLE GAH
...
D789..D7A3    ; LVT
 # Lo  [27] HANGUL SYLLABLE HIG..HANGUL SYLLABLE HIH
```

To turn that into a two-stage table, we first need to do some manipulation to get it into the right form.

We can do that, of course, but there's a time cost, and we would also have to worry about how many intermediate data structures we are using.

Many languages handle this problem using a dedicated script to make their two-stage table structures, then reading some 'digested' form back at runtime. For TEX use, that would probably be better done using a different scripting language: Python has some advantages, but as Lua is the TEX world's standard scripting system, I would favour that. The downside to this approach is you can't use the files from the Unicode Consortium directly, so you have to keep track of your digested set. Also, as in TEX we tend to create formats, and they already *are* digested data dumps, it feels more natural to just read the 'raw' Unicode files as part of format-building, wherever possible.

The outcome of that decision is that there are places where it's easier *not* to use a two-stage table, as we can make a reasonably efficient structure in macros that works 'well enough'. I've done that, for example, for grapheme breaking. There are only about 12 different grapheme breaking classes, and almost all codepoints are in the default one that we *don't* need to record. The raw data are ordered by breaking class, so it's easy to turn that into comma-separated lists of codepoint ranges: one list for each breaking class. Whilst this means that a few codepoints perform slightly less well than others when looking them up,[3] that's acceptable as most of the effort TEX is making here is not the data checking.

## 7 Outlook

Unicode is *the* way that most computer systems work with text data today. Supporting Unicode methods is workable in TEX, even with engines that are fundamentally 8-bit. Over time, more Unicode data will be needed by expl3, and potentially by others, and using the approaches outlined here we can make that available inside TEX runs without needing to look to novel engine extensions.

It's possible that as more data are required, it will be sensible to move from parsing in TEX to parsing in Lua for 'digestion'. But the underlying data structures can remain the same; that is only a question of how best to create them.

◇ Joseph Wright
Northampton, United Kingdom
`joseph dot wright (at) morningstar2.co.uk`

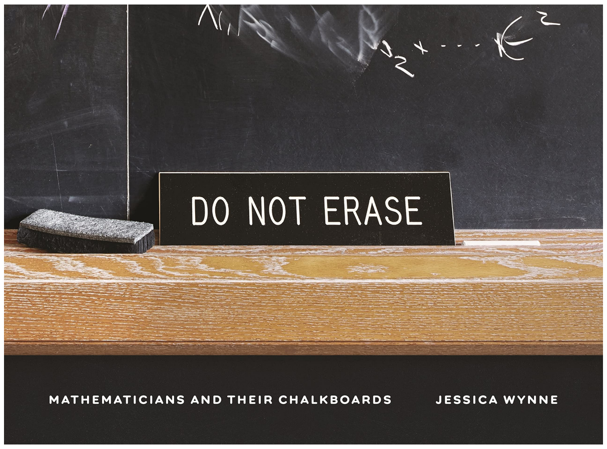
◇ LATEX Project Team
`https://latex-project.org`

[3] Hangul characters: these have the most complex breaking behaviour.

## Book review: *Do Not Erase: Mathematicians and Their Chalkboards*, by Jessica Wynne

Jim Hefferon

Jessica Wynne, *Do Not Erase: Mathematicians and Their Chalkboards*. Princeton University Press, USA, 2021, 240 pp., hardcover, US$35, ISBN 9780691199221.



This book exhibits an art that is small but that is very important to many of us: the handwritten work of mathematicians.

The author introduces the project by describing some friends, "Amie and Benson are theoretical or 'pure' mathematicians ... One day on the Cape, I watch Benson work at his dining room table. ... For several hours, he sits, thinking, creating, jotting down the occasional note. It looks like he has a secret. ... I feel like he is creating something so expansive and beautiful it is beyond words, something that exists only in his head. When I ask him to explain what he is working on, he pauses, appears to struggle for the right words, and replies, simply, 'No. I can't.'"

Later the author also says, "I have always used my camera as a way to understand and explore the world." So while this book is subtitled *Mathematicians and their chalkboards*, another good way to understand it is: glimpses into the works of mathematicians through their chalkboards.

This is a coffee table book. It is beautiful, with typography that is understated yet powerful. It is not coffee table-sized but it has the same feel in that you shouldn't read it, you should browse it. Open it anywhere and you see a two-page spread about one mathematician. Most are research mathematicians, although a few do not fit that description perfectly. Even-numbered pages have a brief biography and an essay by that mathematician reflecting on their work. They write about their perception of beauty, or perhaps a bit about their career and what drew

doi.org/10.47397/tb/44-1/tb136reviews-wynne

them in the direction that they went. It is not the usual thing for mathematicians to put in writing.

Odd-numbered pages show that person's chalkboard. Each picture is different than the others and all are visually interesting.

These are not necessarily candid shots. The author says, "I ask the mathematicians to write or draw whatever they want on their boards. (Often I end up shooting whatever is already on the board — usually something they are currently working on.)" So the subject had the option to put on the board what they want to share. That's wise, if only because for instance my board would contain a grocery list, along with some passwords.

Some boards are messy, some are spare. Some are covered with formulas, while others focus on a figure or two.

I'll take Gilbert Strang as an example. There is a five-sentence biography and his essay focuses on what he is most popularly famous for, his lectures and book on Linear Algebra. His blackboard is taken straight from that class, with matrices and matrix equations hard at work.

The author asserts, "Despite technological advances (such as the creation of computers), chalk on a board is still how most mathematicians choose to work. As musicians fall in love with their instruments, mathematicians fall in love with their boards — the shape, the texture, the quality of the special Japanese Hagoromo chalk." While I don't know that this is completely true, since plenty of people prefer whiteboards, or paper, or tablets, perhaps it doesn't matter. Certainly chalk gives the pictures a theme. Certainly also many of us agree with Sun-Yung Alice Chang who says, "Despite the computer age we live in, the type of talks I enjoy the most are still those in which the speaker writes on the blackboard line by line and explains his or her thoughts."

There are many books about mathematics filled with beautiful graphics. Searching for phrases such as 'mathematics art' or 'mathematics beauty' will produce a list. They often have lots of drawings done with computers, such as fractals, and these can be stunning as well as fascinating. However, as with really high-end natural science illustrations, often somehow the hand-crafted ones are more compelling and show better what it is that the viewer needs to see.

This book also fits with another tradition, ones that reflect on a life in mathematics. Many of these are biographies but there are some that like this one touch on a number of mathematicians, sampling broadly rather than deeply. Two familiar and excellent ones of this type are *Mathematical People* and *More Mathematical People*. One that is recent enough that some readers may not yet have seen it is *Mathematicians: An Outer View of the Inner World*, published by the AMS in 2018.

Alec Wilkinson's afterword is a powerful meditation, "These photographs ... typify the mathematician's historic engagement with beauty." Strictly speaking this book doesn't have to do with TeX — for instance, there is no mathematical typography in the typeset material — but it is about conveying mathematics and about beauty. He closes by saying, "Each of these elegant photographs preserves a detail in the canvas of rigorous human thought."

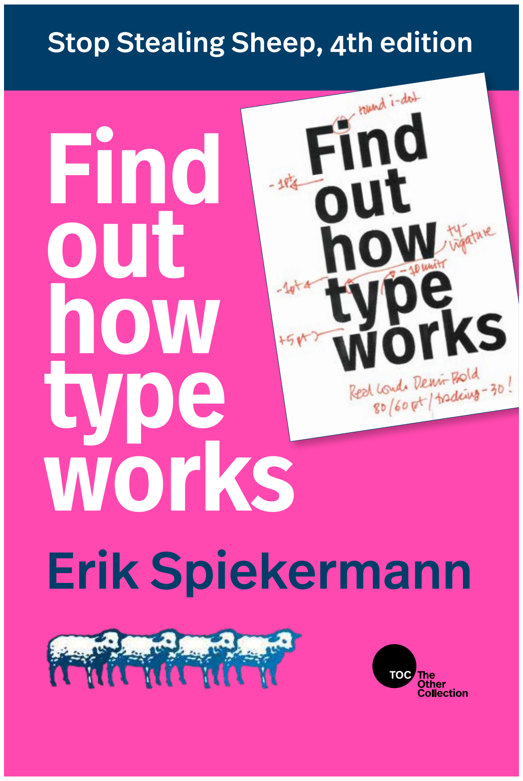
*Do Not Erase* is an excellent choice as a gift for a budding mathematician, to communicate what the life of a mathematician is like, especially that of a researcher. For the same reason it is also a great choice for either an institutional or departmental library. The striking beauty draws the reader in and the essays hold them.

◇ Jim Hefferon
`jhefferon (at) smcvt dot edu`

## Book review: *Stop Stealing Sheep & Find out how type works*, by Erik Spiekermann

John D Lamb

Erik Spiekermann, *Stop Stealing Sheep & Find out how type works*, 4th edition. The Other Collection, 2022, 231 pp., softcover, ISBN 978-3-949164-03-3. Available from `https://fonts.google.com/knowledge/stop_stealing_sheep.pdf` at no cost.



From the start, TEX was designed to set type well. *Stop Stealing Sheep* is a book that explains why that matters. First published in 1993, the book is hardly new. What is new is that the author, Erik Spiekermann, with the help of Google Fonts, has made the book available under a Creative Commons licence, CC BY-ND 4.0 (`creativecommons.org/licenses/by-nd/4.0`).

Spiekermann has written a delightful, readable guide to the world of type. He shows us what type is, how it works, how it makes us feel and how we might choose it for different situations. He explains different varieties of type and how they interact. He shows us how type works at different sizes with tracking, kerning, spacing between words and lines, grids, and the layout of the page. And he builds all of this into a book that more than anything else illuminates his concepts. The left page of each pair is a full-page image that illustrates some concept on the right. And these text pages are filled, but not cluttered, with sidebars, illustrations and marginal notes, all exemplifying how type can be used well. The chapters are short and can be read independently, which makes the book one you can browse or dip into whenever you like.

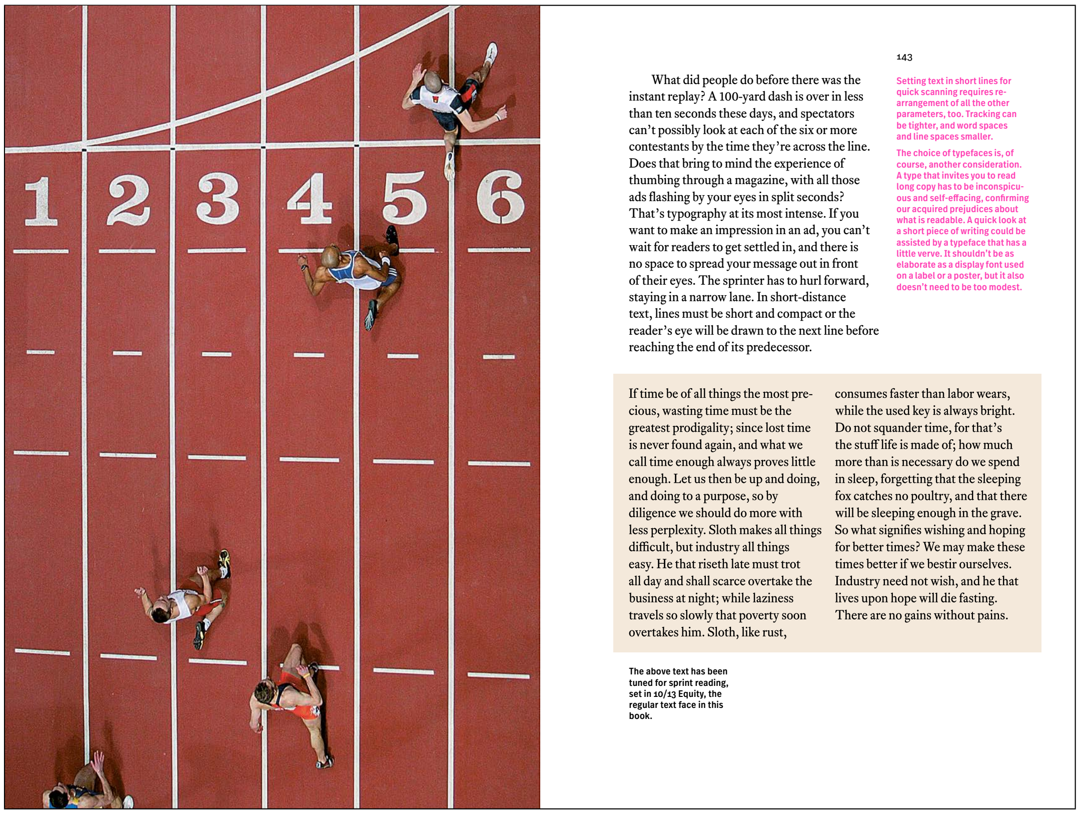
The book's terse, lively style means that it is not a detailed guide on how to choose typefaces (or fonts) or how to decide on such things as column width, line spacing and page layout. For these, I would suggest the book by Williams [1] for a more gentle introduction, or Bringhurst [2] for a comprehensive treatise. But this doesn't mean you won't find much helpful material in the present book.

First, it is worth reading because it helps you understand the value of many of the design choices of TEX. Compared to most word processors, TEX and its variants make it difficult to change typeface whenever you like. Although you can do more than this, the easy option is to choose a package giving you a set of families of typefaces to be used together: one roman, one sans serif and one monospaced. And you don't easily change type sizes arbitrarily. You can use boldface or italics, but underlining is not so easy. On the other hand TEX has hyphenation switched on by default, and goes to great lengths to get it correct, while word processors usually have it switched off and hyphenate more crudely. TEX also uses different spacing between words depending on whether they end sentences or not (by default), produces ligatures by default and allows fine control over spacing and kerning. Spiekermann explains why these and other design choices matter and illustrates how they work to produce more readable, legible and beautiful type.

Second, if you don't want or need detailed type information, *Stop Stealing Sheep* gives you some sensible ideas about the choices you still might make. When should you use `\raggedright` or `flushleft`? Is it a good idea or not to put lines in a table? When should you indent paragraphs or put space between them? When is it better to use two columns rather than one? And why is ALL CAPS not usually a good idea?

While it introduces you to sensible design choices, *Stop Stealing Sheep* is not a book about TEX or any of its variants. Indeed, it mentions neither these nor any composition software, but notes in passing a few other programs such as Adobe Illustrator. So, if you wish to apply the ideas, you will need to look to other sources such as [3] or the various online guides. To use colors (Chapter 3) you might investigate the `color` or `xcolor` packages. The `fontspec` package is

doi.org/10.47397/tb/44-1/tb136reviews-spiekermann



143

What did people do before there was the instant replay? A 100-yard dash is over in less than ten seconds these days, and spectators can't possibly look at each of the six or more contestants by the time they're across the line. Does that bring to mind the experience of thumbing through a magazine, with all those ads flashing by your eyes in split seconds? That's typography at its most intense. If you want to make an impression in an ad, you can't wait for readers to get settled in, and there is no space to spread your message out in front of their eyes. The sprinter has to hurl forward, staying in a narrow lane. In short-distance text, lines must be short and compact or the reader's eye will be drawn to the next line before reaching the end of its predecessor.

Setting text in short lines for quick scanning requires re-arrangement of all the other parameters, too. Tracking can be tighter, and word spaces and line spaces smaller.

The choice of typefaces is, of course, another consideration. A type that invites you to read long copy has to be inconspicuous and self-effacing, confirming our acquired prejudices about what is readable. A quick look at a short piece of writing could be assisted by a typeface that has a little verve. It shouldn't be as elaborate as a display font used on a label or a poster, but it also doesn't need to be too modest.

If time be of all things the most precious, wasting time must be the greatest prodigality; since lost time is never found again, and what we call time enough always proves little enough. Let us then be up and doing, and doing to a purpose, so by diligence we should do more with less perplexity. Sloth makes all things difficult, but industry all things easy. He that riseth late must trot all day and shall scarce overtake the business at night; while laziness travels so slowly that poverty soon overtakes him. Sloth, like rust, consumes faster than labor wears, while the used key is always bright. Do not squander time, for that's the stuff life is made of; how much more than is necessary do we spend in sleep, forgetting that the sleeping fox catches no poultry, and that there will be sleeping enough in the grave. So what signifies wishing and hoping for better times? We may make these times better if we bestir ourselves. Industry need not wish, and he that lives upon hope will die fasting. There are no gains without pains.

The above text has been tuned for sprint reading, set in 10/13 Equity, the regular text face in this book.

**Figure 1**: A page spread (pp. 142–143) from *Stop Stealing Sheep*.

helpful for font selection with OpenType (Chapter 6), while the `geometry` package will help design the page layout (Chapter 8) and `textpos` is a possibility if you want a detailed grid layout (also Chapter 8).

You may also investigate the `unicode-math` package if you want to use OpenType with mathematics. But, to be clear, *Stop Stealing Sheep* is not a book about typesetting mathematics. Indeed, it mentions nothing about mathematics beyond illustrating the various possibilities for setting figures — tabular and proportional, lining and oldstyle. But even here it should be useful if you produce tables of numbers or more readable equations.

In summary, then, while this is not a book about TEX, it is one that I would recommend to anyone who uses TEX.

First, it is an excellent introduction to what type is, how it works and how it can be used. The fourth edition covers many of the features of OpenType, which is becoming the standard for any application that uses type. Whether you want to use OpenType or just use type well, *Stop Stealing Sheep* has much to offer, without excessive technical detail.

Second, TEX remains ahead of any current word processor in its ability to make the features of type available. *Stop Stealing Sheep* will give you a good idea of what these features are. While most are already available in TEX or its derivatives (though many are frustratingly absent in word processors), it is likely that they will become even more widely available in future. So, it makes sense to learn what these features are and why you might want to use them.

Finally, while the printed book is certainly good value for money, the online version is, undeniably, unbeatable value for money.

## References

[1] Robin Williams, *The Non-Designer's Type Book*, 2nd edition, Peachpit Press, 2006.

[2] Robert Bringhurst, *The Elements of Typographic Style*, Version 4.3, Hartley & Marks, 2013.

[3] Frank Mittelbach with Ulrike Fischer, *The LATEX Companion*, 3rd edition, Part 1, Addison Wesley, 2023.

◇ John D Lamb
`j.d.lamb (at) johndlamb dot net`

# The Treasure Chest

These are the new packages posted to CTAN (`ctan.org`) from October 2022–April 2023. Descriptions are based on the announcements and edited for extreme brevity.

Entries are listed alphabetically within CTAN directories. More information about any package can be found at `ctan.org/pkg/`*pkgname*.

A few entries which the editors subjectively believe to be especially notable are starred (*); of course, this is not intended to slight the other contributions.

◇ Karl Berry
`https://tug.org/TUGboat/Chest`
`https://ctan.org/topic`

## biblio

`bibcop` in `biblio/bibtex/utils`
Style checker for `.bib` files.

## fonts

`euler-math` in `fonts`
OpenType implementation of Zapf's Euler.

`gelasio` in `fonts`
Support for Eben Sorkin's Gelasio fonts, metric-compatible with Georgia.

## graphics

`emo` in `graphics`
Emoji support for all LaTeX engines.

## graphics/metapost/contrib/macros

`hershey-mp` in `graphics/metapost/contrib/macros`
Support for Hershey font file format.

`mpchess` in `graphics/metapost/contrib/macros`
Draw chess boards and positions.

## graphics/pgf/contrib

`fenetrecas` in `graphics/pgf/contrib`
Display CAS examples.

`outilsgeomtikz` in `graphics/pgf/contrib`
Typeset geometry tools (compass, ...).

`pixelarttikz` in `graphics/pgf/contrib`
Generate pixel art.

`quickreaction` in `graphics/pgf/contrib`
Typeset chemical reactions.

`sacsymb` in `graphics/pgf/contrib`
"Sacred symbols" used with the Orch OR theory of consciousness.

`scrabble` in `graphics/pgf/contrib`
Commands for Scrabble boards.

`tangramtikz` in `graphics/pgf/contrib`
Tangram puzzles.

`tikz-mirror-lens` in `graphics/pgf/contrib`
Spherical mirrors and lenses.

`tikz-nfold` in `graphics/pgf/contrib`
Triple, quadruple, and $n$-fold paths.

`tikzviolinplots` in `graphics/pgf/contrib`
Draw violin plots from data.

## graphics/pstricks/contrib

`egpeirce` in `graphics/pstricks/contrib`
Draw Peirce's existential graph.

`pst-flags` in `graphics/pstricks/contrib`
Draw country flags.

## info

* `drawing-with-metapost` in `info`
Example-based document for drawing technical diagrams with MetaPost; intermediate to advanced.

* `tlc3-examples` in `info/examples`
All examples from *The LaTeX Companion*, third edition. See excerpt in this issue (pp. 77–86). To order the book, see `tug.org/l/tlc3`.

## macros/generic

`crossrefenum` in `macros/generic`
Smart handling of cross-reference ranges; supports LaTeX and ConTeXt.

`expkv-bundle` in `macros/generic`
Expandable key=val implementation and friends.

`pdfmsym` in `macros/generic`
More math symbols, including arbitrarily-extendable accents, arrows, etc.

## macros/latex/contrib

`cleveref-usedon` in `macros/latex/contrib`
Adds forward referencing to `cleveref`.

`cvss` in `macros/latex/contrib`
CVSS scores for IT vulnerabilities.

`elteiktdk` in `macros/latex/contrib`
Thesis template for Hungarian TDK conferences.

`exam-lite` in `macros/latex/contrib`
Simple exam template.

`fistrum` in `macros/latex/contrib`
150 paragraphs of Lorem Fistrum comedian phrases.

`gfdl` in `macros/latex/contrib`
Use the GNU Free Documentation License (GFDL).

`gradient-text` in `macros/latex/contrib`
Decorate text with linear gradient colors.

`hep-reference` in `macros/latex/contrib`
Adjustments for publications in high energy physics.

`hfutexam` in `macros/latex/contrib`
Exam class for the Hefei University of Technology.

`hwemoji` in `macros/latex/contrib`
Unicode emoji support for pdfLaTeX with sequences.

`ibrackets` in `macros/latex/contrib`
Intelligent brackets for open intervals.

`macros/latex/contrib/ibrackets`

`jeuxcartes` in `macros/latex/contrib`
Playing cards from poker or French tarot decks.

`jourcl` in `macros/latex/contrib`
Universal cover letter for journal submissions.

`jwjournal` in `macros/latex/contrib`
Writing personal journals.

`korigamik` in `macros/latex/contrib`
Support for academic projects and lab reports.

`lgrmath` in `macros/latex/contrib`
Use LGR-encoded fonts in math mode.

`maze` in `macros/latex/contrib`
Generate random mazes.

`naive-ebnf` in `macros/latex/contrib`
Typeset EBNF from a plain text notation.

`namedtensor` in `macros/latex/contrib`
Macros for named tensor notation.

`osda` in `macros/latex/contrib`
Open-Source Design Automation proceedings.

`overarrows` in `macros/latex/contrib`
Custom extensible arrows over math expressions.

* `pagelayout` in `macros/latex/contrib`
Declarative desktop publishing with LaTeX.

`pangram` in `macros/latex/contrib`
LaTeX package for testing fonts.

`physics2` in `macros/latex/contrib`
Easier math typesetting.

`pythonimmediate` in `macros/latex/contrib`
Library to run Python code.

`recorder-fingering` in `macros/latex/contrib`
Typeset recorder fingering diagrams.

`resmes` in `macros/latex/contrib`
The measure restriction symbol.

`resolsysteme` in `macros/latex/contrib`
Operate on linear systems with `xint` or `pyluatex`.

`songproj` in `macros/latex/contrib`
Beamer slideshows with song lyrics.

`tangocolors` in `macros/latex/contrib`
Use colors from the Tango color palette.

`tidyres` in `macros/latex/contrib`
Multi-column formal resumes.

`tramlines` in `macros/latex/contrib`
Lines above and below titles, for legal documents in the UK.

`ukbill` in `macros/latex/contrib`
Typeset UK legislation.

`uol-physics-report` in `macros/latex/contrib`
Lab reports at the U. of Oldenburg.

`uvaletter` in `macros/latex/contrib`
Unofficial letterhead for the U. of Amsterdam.

`writeongrid` in `macros/latex/contrib`
Grid creation and text positioning on the lines.

`zennote` in `macros/latex/contrib`
Streamline note-taking process.

---

`macros/latex/contrib/zennote`

### macros/latex/contrib/babel-contrib

`babel-lithuanian` in `m/l/c/babel-contrib`
Babel support for Lithuanian.

---

### macros/luatex/generic

`blopentype` in `macros/luatex/generic`
Basic LuaTeX OpenType handler.

`evangelion-jfm` in `macros/luatex/generic`
Font metric package supporting many advanced CJK features.

`lparse` in `macros/luatex/generic`
Lua module for parsing key-value options.

`lua-tinyyaml` in `macros/luatex/generic`
Tiny YAML (subset) parser in pure Lua.

`tsvtemplate` in `macros/luatex/generic`
Apply templates to tsv (tab-separated value) files.

---

### macros/luatex/latex

* `luacas` in `macros/luatex/latex`
Computer algebra and symbolic computation system within LuaLaTeX. See article in this issue (pp. 94–98).

`luacomplex` in `macros/luatex/latex`
Perform operations on complex numbers.

`luagcd` in `macros/luatex/latex`
GCD computation.

`lualinalg` in `macros/luatex/latex`
Perform linear algebra operations.

`luamaths` in `macros/luatex/latex`
Perform standard math operations.

`luamodulartables` in `macros/luatex/latex`
Generate modular addition/multiplication tables.

`luaoptions` in `macros/luatex/latex`
Option handling for LuaLaTeX packages.

`luaset` in `macros/luatex/latex`
Perform set operations.

`scikgtex` in `macros/luatex/latex`
Annotate contributions using XMP metadata.

---

### macros/unicodetex/latex

`alchemist` in `macros/unicodetex/latex`
Typeset alchemy and astrological symbols from Unifont

---

### macros/xetex/plain

`unimath-plain-xetex` in `macros/xetex/plain`
OpenType math support in plain XeTeX.

---

### support

`digestif` in `support`
Editor plugin and language for LaTeX, plain TeX, ConTeXt and Texinfo.

* `texfindpkg` in `support`
Install TeX packages via filenames or command/environment names. Supports TL and MiKTeX.

# Abstracts

## *Die TEXnische Komödie* 4/2022–1/2023

*Die TEXnische Komödie* is the journal of DANTE e.V., the German-language TEX user group (`dante.de`).

### Die TEXnische Komödie 4/2022

VOLKER RW SCHAA, Protokoll der 64. Mitgliederversammlung von DANTE am 25. Juni 2022 in Magdeburg [Minutes of the 64. DANTE meeting held on June 25, 2022, in Magdeburg]; pp. 6–13
Minutes of the meeting.

DORIS BEHRENDT, Bericht der Schatzmeisterin für das Jahr 2021 [Treasurer's report for the financial year 2021]; pp. 13–17
Report of the Treasurer for the financial year 2021

MARIO HAUSTEIN AND MAREI PEISCHL, Bericht der Rechnungsprüfer zu den Jahren 2020 und 2021 [Report of the internal auditors for the financial years 2020 and 2021]; pp. 17–22
Report of the internal auditors for the financial years 2020 and 2021.

MAREI PEISCHL, `\dante_tutorial:nn{expl3}{2022}`; pp. 23–36
A tutorial on the new `expl3` syntax. (English translation published in this issue, 87–93.)

HENNING HRABAN RAMM, ConTEXt – bildschön! [ConTEXt — beautiful graphics!]; pp. 37–46
A tutorial on how to handle graphics in ConTEXt.

HENNING HRABAN RAMM, ConTEXt kurz notiert! [ConTEXt News]; pp. 46–49
News regarding ConTEXt.

GÖTZ SCHNELL, Eindrücke vom ConTEXt-Meeting 2022 [Impressions from the ConTEXt Meeting 2022]; pp. 50–54
What happened at the 2022 ConTEXt meeting.

JÜRGEN FENN, Neue Pakete auf CTAN [New packages on CTAN]; pp. 57–62
List of new packages on CTAN.

### Die TEXnische Komödie 1/2023

KENO WEHR, LATEX und Schulphysik 1: Größen und Einheiten [LATEX and Physics in School 1: Dimensions and Units]; pp. 7–16
On the typographical rules for formulas and units. We also present the `schulma-physik` package.

HENNING HRABAN RAMM, Buchumschläge mit ConTEXt [Bookcovers using ConTEXt]; pp. 16–25
A tutorial on how to design bookcovers in ConTEXt.

HENNING HRABAN RAMM, ConTEXt kurz notiert! [ConTEXt news]; pp. 25–27
News regarding ConTEXt.

FRANK MITTELBACH, Das LATEX Tagged PDF Project — Status und Fortschritte [The LATEX Tagged PDF Project — A status and progress report]; pp. 28–39
[Published in *TUGboat* 43:3; translated by Thomas Demmig.]

ULRIKE FISCHER, FRANK MITTELBACH, XMP-Metadaten in LATEX einfügen [Adding XMP metadata in LATEX]; pp. 39–49
[Published in *TUGboat* 43:3; translated by Thomas Demmig.]

FRANK MITTELBACH, LATEX News, issue 36, November 2022; pp. 49–56
[Published in *TUGboat* 43:3; translated by Thomas Demmig.]

JÜRGEN FENN, Neue Pakete auf CTAN [New packages on CTAN]; pp. 56–61
List of new packages on CTAN.

[Received from Uwe Ziegenhagen.]

## *La Lettre GUTenberg* 47–49, 2022–2023

*La Lettre GUTenberg* is a publication of GUTenberg, the French-language TEX user group (`gutenberg-asso.org`); published online at `publications.gutenberg-asso.fr/lettre`.

As of issue #49, *La Lettre GUTenberg* publishes its source code. This code is available by clicking on links located in the journal itself, at the end of each article.

### *La Lettre GUTenberg* #47

Published November 2, 2022. Special issue about the bylaws submitted to the members of the association for a vote, which took place on November 12, 2022.

PATRICK BIDEAULT, Éditorial [Editorial]; p. 1

FLORA VERN, Présentation de la proposition de nouveaux statuts [Presentation of the proposed new bylaws]; pp. 3–4

Proposition de statuts pour GUTenberg [Proposal of bylaws for GUTenberg]; pp. 5–13
The new bylaws were written collectively by all interested members, via public git repository and mailing list.

### *La Lettre GUTenberg #48*

Published December 2, 2022.

Patrick Bideault, Éditorial [Editorial]; pp. 1–3

François Druel, Procès verbal de l'assemblée générale extraordinaire du 12 novembre consacrée à la modification de nos statuts [Minutes of the extraordinary general assembly of November 12 devoted to the modification of our bylaws]; pp. 4–5

Maxime Chupin, Résultat du vote des nouveaux statuts de l'association [Result of the vote on the new bylaws of the association]; p. 6
GUTenberg's new bylaws have been adopted.

Flora Vern, Yvon Henel, Rapport financier pour l'année 2021 [Financial report for the year 2021]; pp. 6–7

Maxime Chupin, Bilan moral : janvier 2021 — décembre 2022 [Annual activity report: January 2021–December 2022]; pp. 8–14

Journée et assemblée générale 2022 [GUTenberg Day and General Assembly 2022]; pp. 14–16
The day's program includes three lectures, by Cédric Pierquet, François Pantigny and Antoine Missier.

Maxime Chupin, Nénufar — Collection de classiques mis en page avec LuaLaTeX [Nénufar — Collection of classics formatted with LuaLaTeX]; pp. 16–18

Patrick Bideault, Denis Bitouzé, Maxime Chupin, Yvon Henel, Et maintenant, une bonne *vieille* veille technologique ! [Technology watch]; pp. 18–22
24 new CTAN packages, October–November 2022.

Patrick Bideault, La fonte de ce numéro : *Plex* [This issue's font: Plex]; pp. 22–24

Patrick Bideault, À propos des fontes que nous utilisons [About the fonts in use]; pp. 24–26
A short article about the desired specifications of the fonts in use in the bulletin.

Patrick Bideault, En bref [At a glance]; pp. 26–28
Short news about capitals, acronyms, fonts, footnotes and more.

### *La Lettre GUTenberg #49*

Published January 23, 2023.

Patrick Bideault, Éditorial [Editorial]; pp. 1–2

Patrick Bideault, Denis Bitouzé, Maxime Chupin, François Druel, Procès verbaux [Reports of the board's meetings]; pp. 2–10

Maxime Chupin, Les vidéos de la Journée GUTenberg 2022 sont en ligne ! [The videos of the GUTenberg 2022 conferences are online]; pp. 10–12

Patrick Bideault, Denis Bitouzé, Maxime Chupin, Yvon Henel, Et maintenant, une bonne *vieille* veille technologique ! [Technology watch]; pp. 12–17
30 new CTAN packages, December 2022–January 2023.

Patrick Bideault, À propos des documentations de packages et des moyens de contacter leurs auteurs [About package documentation and how to contact their authors]; p. 18
Advocacy about documentation formats and ways to contact package authors.

Maxime Chupin, Sondage sur les utilisateurs francophones de (La)TeX [Survey of French-speaking users]; pp. 18–19

Denis Bitouzé, Maxime Chupin, Passer à la définition de commandes de LaTeX3 [Switch to the LaTeX3 macro definition commands]; pp. 19–24

Denis Bitouzé, Maxime Chupin, La fonte du numéro : Fira Sans [This issue's font: Fira Sans]; pp. 25–28

Patrick Bideault, Maxime Chupin, En bref [At a glance]; pp. 28–29
Short news about a website and a nice suggestion.

Yvon Henel, Rébus, cacotypographie et archéotypographie [A rebus and considerations on cacotypography and archeotypography]; pp. 29–30

Patrick Bideault, Un beau moment de lecture [A nice reading moment]; pp. 30–31
About Gerard Unger's book *While You're Reading.*

[Received from Patrick Bideault.]

## *Zpravodaj* 2022/1–4

*Zpravodaj* is the journal of C̲S̲TUG, the TEX user group oriented mainly but not entirely to the Czech and Slovak languages. The full issue can be downloaded at `cstug.cz/bulletin`.

Petr Sojka, Úvodník [Introductory word]; pp. 1–2

Go forth and participate in CSTUG to make the bright future of TEX & Friends a reality! *You can!*

Petr Olšák, cropmarks.tex – makra na tvorbu ořezových značek [`cropmarks.tex` — Macros for creating crop marks]; pp. 4–10

The `cropmarks.tex` package included in the `olsak-misc` bundle of packages is presented here. It enables adding configurable crop marks to any PDF file (no matter what software created it). The macro package is based on plain TEX and works in OpTEX too. The special crop marks needed for impositions are also supported, and this example is shown in the article in detail.

A summary of other `olsak-misc` macro files is given in the last section. Moreover, the macros are documented at the end of each file.

Tereza Vrabcová, Digitální archivace Zpravodaje CSTUGu [Digital archival of the *CSTUG Bulletin*]; pp. 11–17

When a person is proud of their creation, they want to share it with as many people as possible. Therefore, it cannot be a surprise that CSTUG took the opportunity to digitally archive the *CSTUG Bulletin* and make its articles accessible to the general public with the help of the Czech Digital Mathematics Library (DML-CZ). This article introduces the technical solution and the results of this digitization, along with some interesting statistics that were discovered during the process. It is adapted from the author's talk at the general assembly of CSTUG on May 14, 2022.

Denis Roegel, Romantika v METAPOSTu po francouzsku: líbající se kružnice [Kissing circles: A French romance in METAPOST]; pp. 18–34

Published in *TUGboat* 26:1.

Vít Novotný, Vysokoúrovňové jazyky pro TeX [High-level languages for TEX]; pp. 35–48

TEX is the assembly language of digital typesetting, which requires advanced programming skills from authors and designers, and which provides few high-level abstractions to programmers. In this article, I introduce selected markup, programming, and style-sheet languages for TEX, which enable the division of labor between authors, programmers, and designers, and which simplify the process of electronic document preparation. The article is a transcription of my invited talk at the general assembly of CSTUG on May 14, 2022.

Max Chernoff, Automatically removing widows and orphans with `lua-widow-control`; pp. 49–76

Published in *TUGboat* 43:1; now updated.

Peter Wilson, Mělo by to fungovat XII [It might work XII]; pp. 77–88

Published in *TUGboat* 32:1 as "Glisterings".

[Received from Vít Novotný.]

## Advertisements

**TEXnology Inc.**

Amy Hendrickson
57 Longwood Ave. #8
Brookline, MA 02446
+1 617-738-8029
Email: amyh (at) texnology.com
Web: https://texnology.com

Full time LATEX consultant for more than 30 years; have worked for major publishing companies, leading universities, and scientific journals. Our macro packages are distributed on-line and used by thousands of authors. See our site for many examples: texnology.com.

- **LATEX Macro Writing:** Packages for books, journals, slides, posters, e-publishing and more; Sophisticated documentation for users.
- Design as well as LATEX implementation for e-publishing, print books and journals, or specialized projects.
- Data Visualization, database publishing.
- Innovative uses for LATEX, creative solutions our speciality.
- LATEX Training, customized to your needs, on-site or via Zoom. See https://texnology.com/train.htm for sample of course notes.

Call or send email: I'll be glad to discuss your project with you.

doi.org/10.47397/tb/44-1/tb136zprav

# TEX Consultants

The information here comes from the consultants themselves. We do not include information we know to be false, but we cannot check out any of the information; we are transmitting it to you as it was given to us and do not promise it is correct. Also, this is not an official endorsement of the people listed here. We provide this list to enable you to contact service providers and decide for yourself whether to hire one.

TUG also provides an online list of consultants at tug.org/consultants. If you'd like to be listed, please visit that page.

**Dangerous Curve**
Email: typesetting (at) dangerouscurve.org

Typesetting for over 40 years, we have experience in production typography, graphic design, font design, and computer science, to name a few things. One DC co-owner co-authored, designed, and illustrated a TEX book (*TEX for the Impatient*).

We can ▪ convert your documents to LATEX from just about anything ▪ type up your handwritten pages ▪ proofread, copyedit, and structure documents in English ▪ apply publishers' specs ▪ write custom packages and documentation ▪ resize and edit your images for a better aesthetic effect ▪ make your mathematics beautiful ▪ produce commercial-quality tables with optimal column widths for headers and wrapped paragraphs ▪ modify bibliography styles ▪ make images using TEX-related graphic programs ▪ design programmable fonts using METAFONT ▪ and more! (Just ask.)

Our clients include high-end branding and advertising agencies, academics at top universities, leading publishers. We are a member of TUG, and have supported the GNU Project for decades (including working for them). All quote work is complimentary.

**Latchman, David**
2005 Eye St. Suite #6
Bakersfield, CA 93301
+1 518-951-8786
Email: david.latchman (at) texnical-designs.com
Web: www.texnical-designs.com

LATEX consultant specializing in the typesetting of books, manuscripts, articles, Word document conversions as well as creating the customized LATEX packages and classes to meet your needs. Contact us to discuss your project or visit the website for further details.

**Veytsman, Boris**
132 Warbler Ln.
Brisbane, CA 94005
+1 703-915-2406
Email: borisv (at) lk.net
Web: www.borisv.lk.net

TEX and LATEX consulting, training, typesetting and seminars. Integration with databases, automated document preparation, custom LATEX packages, conversions (Word, OpenOffice etc.) and much more.

I have about two decades of experience in TEX and three decades of experience in teaching & training. I have authored more than forty packages on CTAN as well as Perl packages on CPAN and R packages on CRAN, published papers in TEX-related journals, and conducted several workshops on TEX and related subjects. Among my customers have been Google, US Treasury, FAO UN, Israel Journal of Mathematics, Annals of Mathematics, Res Philosophica, Philosophers' Imprint, No Starch Press, US Army Corps of Engineers, ACM, and many others.

We recently expanded our staff and operations to provide copy-editing, cleaning and troubleshooting of TEX manuscripts as well as typesetting of books, papers & journals, including multilingual copy with non-Latin scripts, and more.

**Warde, Jake**
90 Resaca Ave.
Box 452
Forest Knolls, CA 94933
+1 650-468-1393
Email: jwarde (at) wardepub.com
Web: myprojectnotebook.com

I have been in academic publishing for 30+ years. I was a linguistics major at Stanford in the mid-1970s, then started a publishing career. I knew about TEX from editors at Addison-Wesley who were using it to publish beautifully set math and computer science books.

Long story short, I started using LATEX for exploratory projects (see website above). I have completed typesetting projects for several journal articles. I have also explored the use of multiple languages in documents extensively. I have a strong developmental editing background in STEM subjects. If you need assistance getting your manuscript set in TEX I can help. And if I cannot help I'll let you know right away.

## TUG 2023 election report

Nominations for TUG President and the Board of Directors in 2023 have been received and validated. Because there is a single nomination for the office of President and because there are not more nominations for the Board of Directors than there are open seats, there is no requirement for a ballot this election.

For President, Arthur Rosendahl was nominated. As there were no other nominees, he is duly elected and will serve for a two-year term.

For the Board of Directors, the following individuals were nominated:

Barbara Beeton, Max Chernoff, Ulrike Fischer, Jim Hefferon, Tom Hejda, Jérémy Just, Norbert Preining, Boris Veytsman.

As there were not more nominations than open positions, all the nominees are duly elected to a four-year term. Thanks to all for their willingness to serve.

Terms for both President and members of the Board of Directors will begin at the Annual Meeting.

Board member Paulo Cereda has decided to step down. Paulo was appointed to the Board in 2021 to fill a position that remained open after the election. His dedication and service to the community are gratefully acknowledged.

Election statements by all candidates are given below. They are also available online, along with announcements and results of previous elections.

◇ Karl Berry
for the Elections Committee
tug.org/election

### Arthur Rosendahl (né Reutenauer)

(Candidate for TUG President.)

*Biography*: I first came across TEX as a mathematics student 25 years ago and was instantly hooked. A few years later, my interest in languages gave me another angle to look at TEX and related programs, especially Metafont. I have been involved in some aspects of their development ever since, through packages such as polyglossia and hyph-utf8, and have participated in many conferences to meet with other TEX enthusiasts, and spread the knowledge. I am mostly a ConTEXt user nowadays. I love TEX and its ecosystem for the freedom it gives its users and the taste of beautifully typeset documents.

My involvement with TUG started in 2005 at the annual conference in Wuhan (since then known for other things ... ) and I have been on the board for ten years, the last six of which as vice president. I've been part of the organisation committee of the three online TUG conferences from 2020 to 2022 and am also active in other user groups of the TEX world, as the founding president of the ConTEXt Group, and a board member of GUTenberg, the French-language TEX users group. I was incidentally born on the year TUG was founded.

*Statement*: While TEX itself hasn't changed at all since Don Knuth released version 3 in 1990, almost every single program around it has, as well as the way we use them. The set of users has grown dramatically, as well as their reason for using them, and while one may think that there is less justification for a nonprofit organisation today — when commercial companies such as Stack Exchange and Overleaf have seemingly taken over the stage — I believe that the TEX Users Group is more necessary today than ever, as a gathering place for all the people who give their time, their energy, and not seldom their personal money, so that this beautiful program can long endure.

### Barbara Beeton

(Candidate for TUG Board of Directors.)

*Biography*: For TEX and the TEX Users Group:

- charter member of the TEX Users Group; charter member of the TUG Board of Directors;
- *TUGboat* Editor.

Retired from the American Mathematical Society.

- Staff Specialist for Composition Systems, responsible for documentation and author support;
- STIX representative to the Unicode Technical Committee for adoption of additional math symbols, which were added effective with Unicode 4.0;
- co-author of Unicode Technical Report #25, Unicode Support for Mathematics;

Publications communicating the history of TEX.

- principal author, Communication of Mathematics with TEX, *Visible Language* **50**:2 (2016), 40–51;
- co-author (with Dave Walden and Karl Berry), TEX, A branch of desktop publishing (Parts 1

doi.org/10.47397/tb/44-1/tb136elec

and 2), *IEEE Annals of the History of Computing*, **40**:3 (2018), 78-93, **41**:2 (2019), 29–41.

*Statement*: Although I will be retired from the AMS I intend to continue to be active in TUG, where I have made so many good friends. As the oldest user group in the worldwide TEX community, TUG provides a focus for dedicated TEX users and developers.

I believe there's still a place in the TUG ranks for one of the "old guard", to provide institutional memory when it's appropriate, and cheer on the younger folks who are trying new things.

With support from the members of this wonderful community, I'd like to continue for four more years.

### Max Chernoff

(Candidate for TUG Board of Directors.)

*Biography*: I was a fairly advanced user of Microsoft Word, but about 6 years ago I became frustrated with its clumsy input. I discovered LATEX and I immediately fell in love. All of my free time was quickly replaced by me making new document styles and remaking old documents.

These days, I process most of my documents with ConTEXt, although I occasionally use LATEX and Plain. I am the author of the LuaTEX package lua-widow-control as well as the associated *TUGboat* articles.

I am currently studying Math and Physics as an undergraduate at the University of Calgary.

*Statement*: Despite its remarkable design, TEX is beginning to show its age. The TEX community has put in extensive work to develop modern improvements like LuaTEX, expl3, and BibLATEX/Biber, yet most IDEs/editors, publishers, and tutorials still recommend or require pdfLATEX and BibTEX. These tools still have their place, but I believe that their limitations are a major barrier for newer users. As a TUG director, I would encourage the adoption of LuaTEX and Biber as defaults. I would also support modernisation in general, while maintaining backwards compatibility and avoiding unnecessary changes.

### Ulrike Fischer

(Candidate for TUG Board of Directors.)

*Biography*: I was born in Stuttgart, Germany, and moved first to Switzerland with my parents and later to Bonn where I studied mathematics. I wrote my thesis in a rather arcane branch called model theory using an Atari text processor called Signum. But I did not like to have to place the many sub- and superscripts with a mouse. So when seeing an example LATEX document, I ordered the floppy discs and never looked back.

I found my way into the LATEX groups on the Internet and have always enjoyed and still enjoy answering questions and debugging errors. I also like to help new users to find their way into the TEX world.

Talks about accessibility and tagged PDF at Dante and TUG meetings induced me to write the tagpdf package and to present it to LATEX team members at a workshop at the TUG meeting in Rio de Janeiro in 2018. This got me an invite to join the LATEX Team and to work on the long term project to enable LATEX to create tagged PDF thus ensuring that it will remain an important and relevant document source format. I also helped with the production of the third edition of the *LATEX Companion* which hopefully will be released at end of April.

*Statement*: I joined the TUG board in 2020 on invitation and would be happy to serve four more years ( :-). Stable user groups like TUG can offer long term support and communication means that you don't get in fast changing internet groups. They also can organize in person meetings. The last years have shown to me that this is something that I would really miss. So when the TUG board discussed the next TUG conference, my husband Gert and I offered to be the local organizers in the summer of 2023. It would be really nice to welcome and meet many TUG members and other TEX friends in my hometown Bonn as a TUG board director!

## Jim Hefferon

(Candidate for TUG Board of Directors.)
*Biography*: My biography is that I have been a faculty member in mathematics for many years. I have used TeX and LaTeX since the early 90s for a variety of projects, including professional articles and open source technical books. I helped run CTAN for a decade. I have been on the TUG Board for a number of terms, including terms as Vice President, and I had the privilege of acting as President for a short time.
*Statement*: I would like to continue to work on bringing in new members, both to TUG and to the wider TeX community. Sometimes a group of experienced users could forget the needs of people starting out, and I try to be an advocate, to help keep them in focus.

## Tom Hejda

(Candidate for TUG Board of Directors.)
*Biography*: I earned my master's in engineering in 2012 and my PhD in mathematics and computer science in 2016. Parallel to my studies, I was a technical LaTeX editor for a small journal for over 6 years. After a few years of postdocs, I joined Overleaf in 2019 as a support team member and later as a manager. I also volunteer with the Scouts movement as a manager and a DIYer, and at a local church as an organist.
*Statement*: I would like to join the Board for three reasons.

First, I have been helping run the online TUG meetings and it makes sense to me to continue to offer my help even as we hopefully move back to in-person meetings, with the possibility of running one of the conferences in Prague if there is interest.

Second, at Overleaf, we have a first-hand experience with a lot of LaTeX users and we are most likely the largest entity using TeX Live (our annual deployment of TeX Live to the users and the associated testing have helped with kernel and package debugging before), and I am happy to look for more improvements in this direction.

Third, I would like to offer my experience as a manager in the non-profit/volunteering sector to the TeX community by becoming a member of the Board.

## Jérémy Just

(Candidate for TUG Board of Directors.)
*Biography*: I'm a biologist who studies plant genomes: over the years, I've worked on research projects on rapeseed, wheat, rose, and more recently mosses. As for LaTeX I had my first experience with it in 2000, spontaneously starting to use it to write my reports during my Master of Science. I never stopped.

I've been active in the LaTeX community on Usenet (fr.comp.text.tex) since 2003. There, I discovered the existence of TeX users groups and quickly became a member of GUTenberg, the French-speaking users group. In 2009, willing to help the community, I joined GUTenberg board, and started to serve the association as its treasurer, then as its president. I stayed at GUTenberg until 2022.

Within this association, I took care of the daily administrative stuff, kept the website up to date, set up a CTAN mirror, made the link with TUG and other groups... Since 2016, I'm also the main maintainer of the French-speaking LaTeX FAQ now hosted at `https://www.latex-fr.net/` and I've translated several LaTeX documentations into French, notably the LearnLaTeX website, a tutorial targeting LaTeX beginners (`https://www.learnlatex.org/`).

The past few editions of the yearly TUG conference being held online, that offered me the opportunity to give a hand in their organization.
*Statement*: LaTeX is a wonderful tool for efficiently producing good-looking documents and plots. But one must admit that its learning curve is quite steep. I believe that TeX users groups have a strong role to play in lowering the first step, by encouraging all useful initiatives: writing documentations for all levels, translation of documentations, interactive web sites, video tutorials... As a side-effect, this would very certainly popularize the use of LaTeX in fields where it's not (yet) a habit to use it: biology, humanities ...

### Norbert Preining

(Candidate for TUG Board of Directors.)

*Biography*: I am a mathematician and computer scientist living and working whereever I find a job. After my studies at the Vienna University of Technology, I moved to Tuscany, Italy, for a Marie Curie Fellowship. After another intermezzo in Vienna I have settled in Japan since 2009, as Associate Professor, and since several years now in various companies mostly working on Machine Learning systems.

After years of being a simple user of (LA)TEX, I first started contributing to TEX Live by compiling some binaries in 2001. In 2005, I started working on packaging TEX Live for Debian, which has developed into the standard TEX package for Debian and its derivatives. During EuroBachoTEX 2007, I got (by chance) involved in the development of TEX Live itself, which is now the core of my contribution to the TEX world. Up till now I am continuing with both these efforts.

Furthermore, with my move to Japan I got interested in its typographic tradition and support in TEX. I am working with the local TEX users to improve overall Japanese support in TEX (Live). In this course we managed to bring the TUG 2013 conference for the first time to Japan.

More details concerning my involvement in TEX, and lots of anecdotes, can be found at the TUG interview corner (`tug.org/interviews/preining.html`) or on my web site (`preining.info`).

*Statement*: After many years in the active development and eight years on the board of directors of TUG, I want to continue serving TUG and the wider TEX community.

The challenges I see for TUG remain the same over the years: increase of members and funds, and technical improvement of our software. Promoting TEX as a publishing tool also outside the usual math/CS environment will increase the acceptance of TEX, and by this will hopefully bring more members to TUG.

### Boris Veytsman

(Candidate for TUG Board of Directors.)

*Biography*: I was born in Odesa, Ukraine. After getting my degree in Theoretical Physics I worked in academia and industry, doing research in many areas including physics of fluids, materials science, biophysics, medical physics, air traffic safety, cellular communications, science of science, etc. I also performed TEX consulting for a number of customers from publishers to universities to government agencies to non-profits. My current CV and the list of publications is available at `https://borisv.lk.net/cv/cv.html`.

I have been using TEX since 1994 and have been a TEX consultant since 2005. I published a number of packages on CTAN and papers in *TUGboat*. I have been a Board member in 2010–2016, Vice-President in 2016–2017, and President in 2017–2023. I am an Associate Editor of *TUGboat* and support `https://www.tug.org/books/`. I support TUG accounts at Twitter (`@TeXUsersGroup`) and Mastodon (`@TeXUsersGroup@TechHub.social`).

TEX and friends are my primary tools and the source of my livelihood. I spend most of my working day producing TEX documents. I always felt that I ought to give back to the community: writing packages, editing, mentoring, and serving TUG in different capacities.

*Statement*: I think TEX is an important part of publishing infrastructure, of scientific communications, and of communications in many other areas, from music to literature to philology. It is our job as members of TUG to advocate it, to support it and to coordinate its development. It have been a privilege for me to be a part of this movement. I hope to be able to continue this service.

## TUG financial statements for 2022

Karl Berry, TUG treasurer

The financial statements for 2022 have been reviewed by the TUG board but have not been audited. The totals may vary slightly due to rounding. As a US tax-exempt organization, TUG's annual information returns are publicly available on our web site, below.

### Revenue (income) highlights

Membership dues revenue was down in 2022 compared to 2021; we ended the year with 1,162 paid members, 48 fewer than in 2021. The 2022 online conference had a net gain of over $4,300, due to few expenses and generous donations — thank you! General contributions were substantially increased (74%) due to individual bequests and the generous donation from UK-TUG after their dissolution. Product sales were unusually high due to a major licensing of Lucida to NASA. Overall, 2022 income was up almost 30%.

### Other highlights; the bottom line

Apart from Lucida (per above) and payroll (fulfilling past obligations), other cost categories were about the same.

Our bottom line for 2022 was substantially positive: $23,391.

### Balance sheet highlights

Due to that bottom line, TUG's end-of-year asset total was up around 15% in 2022 compared to 2021.

Committed Funds are reserved for designated projects: LaTeX, CTAN, MacTeX, the TeX development fund, and others (`https://tug.org/donate`). TUG charges no overhead to administer these funds.

The Prepaid Member Income category is member dues that were paid in earlier years for the current year (and beyond). The 2022 portion of this liability was converted into regular Membership Dues in January of 2022. The payroll liabilities are for 2022 state and federal taxes due January 15, 2023.

### Upcoming

We have not changed any rates or fees for 2023, despite increased costs. Worldwide support from members and donations are what allow us to continue, so thank you! As always, we welcome ideas to attract new members.

◇ Karl Berry, TUG treasurer
`https://tug.org/tax-exempt`

**TUG 12/31/2022 (vs. 2021) Revenue, Expense**

| | Dec 31, 22 | Dec 31, 21 |
|---|---|---|
| ORDINARY INCOME/EXPENSE | | |
| Income | | |
| Membership Dues | 76,940 | 79,320 |
| Product Sales | 20,008 | 4,423 |
| Contributions Income | 37,055 | 21,311 |
| Annual Conference | 4,325 | 2,636 |
| Interest Income | 742 | 184 |
| Advertising Income | 375 | 565 |
| Total Income | 139,445 | 108,439 |
| Cost of Goods Sold | | |
| TUGboat Prod/Mailing | (22,639) | (22,053) |
| TUGboat Crossref | (369) | (275) |
| Software Prod/Mailing | (2,818) | (2,391) |
| Members Postage/Delivery | (1,822) | (1,827) |
| Lucida Sales to B&H | (9,595) | (1,675) |
| Member Renewal | (520) | (372) |
| Total COGS | (37,763) | (28,593) |
| Gross Profit | 101,682 | 79,846 |
| Expense | | |
| Contributions made by TUG | | (2,000) |
| Office Overhead | (12,647) | (12,924) |
| Payroll Expense | (71,565) | (64,274) |
| Interest Expense | | (84) |
| Total Expense | (84,212) | (79,282) |
| Net Ordinary Income | 17,470 | 565 |
| OTHER INCOME/EXPENSE | | |
| Prior year adjustment | 5,921 | |
| NET INCOME | 23,391 | 565 |

**TUG 12/31/2022 (vs. 2021) Balance Sheet**

| | Dec 31, 22 | Dec 31, 21 |
|---|---|---|
| ASSETS | | |
| Current Assets | | |
| Total Checking/Savings | 198,499 | 173,601 |
| Accounts Receivable | 2,335 | 395 |
| Total Current Assets | 200,834 | 173,996 |
| LIABILITIES & EQUITY | | |
| Current Liabilities | | |
| Committed Funds | 53,524 | 55,656 |
| Administrative Services | 1,443 | 1,445 |
| Prepaid Member Income | 11,395 | 10,075 |
| Payroll Liabilities | 3,539 | 1,281 |
| Total Current Liabilities | 71,901 | 68,455 |
| Equity | | |
| Unrestricted | 105,542 | 104,977 |
| Net Income | 23,392 | 565 |
| Total Equity | 128,934 | 105,542 |
| TOTAL LIABILITIES & EQUITY | 200,835 | 173,999 |

# Calendar

## 2023

| | |
|---|---|
| Apr 29 – May 3 | BachoTEX 2023, "A model kit. Modeling and implementing text typesetting in TEX and other systems", 28th BachoTEX Conference, Bachotek, Poland. `www.gust.org.pl/bachotex/2023-en` |
| May 10 – 13 | Association Typographique Internationale, ATypI Paris 2023 (hybrid) Paris, France. `atypi.org/conferences-events/atypi-paris-2023` |
| May 20 | GuIT Meeting 2023, 19th Annual Conference, Rome, Italy. `www.guitex.org/home/en/meeting` |
| Jun 1 – 3 | Markup UK, A Conference about XML and Other Markup Technologies, Queen Mary University of London, UK `markupuk.org` |
| Jun 3 | TypeParis23, talks on typography's visible impact; Workshop No. 16, Jun 4: Font Nerdery, Paris, France. `typeparis.com` |
| Jun 26 – 29 | SHARP 2023, "Affordances and Interfaces: Textual Interaction Past, Present and Future", Society for the History of Authorship, Reading & Publishing. Hosted online by the University of Otago, New Zealand. `www.sharpweb.org/main/conferences` |
| Jun 28 – 30 | Twenty-first International Conference on New Directions in the Humanities, "Literary Landscapes: Forms of Knowledge in the Humanities", Sorbonne University, Paris, France. `thehumanities.com/2023-conference` |
| Jul 10 – 14 | Digital Humanities 2023, Alliance of Digital Humanities Organizations, "Collaboration as Opportunity", Graz, Austria. `dh2023.adho.org` |
| Jul 13 | DANTE 65th meeting, Bonn, Germany. `dante.de/veranstaltungen/dante2023` |

| **TUG 2023** | **Bonn, Germany** |
|---|---|
| Jul 13 | Developers' workshop: Tagging PDF documents, 2:00–6:00 PM (space is limited) |
| Jul 13 | Guided walking tour of downtown Bonn, 3:00–6:00 PM |
| Jul 13 | Opening reception, 7:00–8:00 PM |
| Jul 14 – 16 | The 44th annual meeting of the TEX Users Group. Presentations covering the TEX world `tug.org/tug2023` |
| Jul 16 | Excursion: boat trip to the Seven Hills, 2:15 PM |
| Jul 17 | Guided walking tour of downtown Bonn, 9:30 AM–12:30 PM |

| | |
|---|---|
| Jul 23 | **Final papers due for TUG 2023 proceedings** |
| Jul 31 – Aug 4 | Balisage: The Markup Conference (virtual). `www.balisage.net` |
| Aug 6 – 10 | SIGGRAPH 2023, 50 years of SIGGRAPH, Los Angeles, California. `s2023.siggraph.org` |
| Aug 22 – 25 | 23rd ACM Symposium on Document Engineering, Limerick, Ireland. `doceng.org/doceng2023` |
| Sep 5 | The Updike Prize for Student Type Design, application deadline, 5:00 p.m. EST. Providence Public Library, Providence, Rhode Island. `prov.pub/updikeprize` |
| Sep 10 – 16 | 17th International ConTEXt Meeting, Prague–Sibřina, Czech Republic. `meeting.contextgarden.net/2023` |

## 2024

| | |
|---|---|
| Feb 4 – 7 | CODEX IX, Richmond, California. `www.codexfoundation.org` |

*Status as of 5 April 2023*

For additional information on TUG-sponsored events listed here, contact the TUG office (+1 503 223-9994, fax: +1 815 301-3568, email: `office@tug.org`). For events sponsored by other organizations, please use the contact address provided.

User group meeting announcements are posted at `tug.org/meetings.html`. Interested users can subscribe and/or post to the related mailing list, and are encouraged to do so.

Other calendars of typographic interest are linked from `tug.org/calendar.html`.

**TUGBOAT** Volume 44 (2023), No. 1